technische universität
dortmund

Bachelorarbeit

Deep Submodular Autoencoder für Datenzusammenfassung

Minsu So
Oktober 2021

Gutachter:
Prof. Dr. Katharina Morik
Sebastian Buschjäger (M.Sc.)

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz (LS VIII)
https://www-ai.cs.tu-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Hintergrund

Big Data ist eine sogenannte Überflutung von sich regelmäßig verändernden Daten, die komplex und nicht in einem bestimmtem Format festgelegt sind. Mit ihnen können neue Erkenntnisse, beispielsweise für die Entscheidungsverfahren oder Prozessoptimierungen, gewonnen werden. Trotz dieses Vorteils ergeben sich gewisse Herausforderungen, die auch in der nahen Zukunft eine größere Rolle spielen werden. Nicht nur bringen sie Schwierigkeiten für die Verarbeitung und Analyse mit, sondern benötigen auch großen Speicherplatz und Zeitaufwand. Außerdem führt der wachsende Trend von Big Data in Zukunft zu einem größeren Problem in Bezug auf den Ressourcenverbrauch und die Umwelt.

Eine mögliche Lösung zur Überwindung dieser Probleme besteht darin, die Daten so zusammenzufassen, dass sie weniger Speicherplatz, Zeit und Ressourcen bei der Verarbeitung benötigen und für Menschen übersichtlicher werden. Eine Zusammenfassung ist eine kompakte, aber informative Beschreibung eines Datensatzes. Dabei soll eine möglichst kleine „repräsentative“ Teilmenge gefunden werden, die die wesentlichen Inhalte der Ursprungsmenge weiterhin enthält. Es existieren bereits mehrere Ansätze um eine Zusammenfassung zu erstellen, wie z.B. „Clustering“, „Sampling“ und „Compression“[26].

Diese Arbeit stellt die Maximierung von submodularen Funktionen als ein weiteres Verfahren zur Erstellung einer Datenzusammenfassung vor, die auf Grund ihrer Eigenschaft eine optimale Zusammenfassung mit einer theoretischen, festen Worse-Case Garantie liefert [11]. Dabei dienen diese Funktionen zur Messung der Repräsentativität einer bestimmten Teilmenge der Daten. Außerdem kann aufgrund der *Diminishing-Returns* Eigenschaft der Submodularität eine kompaktere Teilmenge gefunden werden.

Da die submodularen Funktionen auf die Berechnung der Ähnlichkeit der Beobachtungen basieren, ist ihre Performanz abhängig von der Repräsentation der gegebenen Daten. Folglich besteht ein Wunsch für komplexe Datensätze, die Daten von ihrem komplexen, hoch dimensionalen Raum auf den eingebetteten Merkmalsraum zu führen.

1.2 Ziel der Arbeit

In dieser Arbeit werden zwei *Deep Learning*-basierte Ansätze zur Erstellung einer Datenzusammenfassung für hochdimensionale Daten genauer untersucht.

- Das *Deep Embedded Clustering* [60] wurde von Xie et al. vorgestellt, um die Clusteringanalyse in der hohen Dimension zu ermöglichen. Es versucht die Merkmalsrepräsentationen und die Clusterzuordnungen der urprünglichen Daten zu lernen. Hierbei stellt sich die Frage, inwiefern das Verfahren zur Erstellung einer Zusammenfassung geeignet ist und was für Schwierigkeiten es aufweist.
- Der *Deep Submodular Autoencoder* ist ein Verfahren, welches ihren Fokus insbesondere auf die Erstellung einer Zusammenfassung für hoch dimensionale Daten setzt. Es handelt es sich um einen erweiterten Ansatz des Deep Embedded Clusterings. Der wesentliche Unterschied ist die Einführung einer submodularen Funktion für das Lernen der Clusterzuordnung. Hierbei stellt sich die Frage, wie die Implementierung dieses Verfahrens genau umzusetzen ist und was dabei besonders geachtet werden muss.

Ziel dieser Arbeit ist die Auseinandersetzung mit den Fragestellungen beider Ansätze.

1.3 Aufbau der Arbeit

Im Kapitel 2 werden Grundlagen zu submodularen Funktionen und zum submodularen Maximierungsproblem eingeführt. Insbesondere wird ein Beispiel einer submodularen Funktion im Abschnitt 2.2 und ein Approximationsalgorithmus für das Maximierungsproblem im Abschnitt 2.3 präsentiert. Um eine niedrig-dimensionale Repräsentation der Daten mit minimalen Informationsverlust zu berechnen, führt das Kapitel 3 das Konzept des Autoencoders ein. Hierbei soll ebenso darauf eingegangen werden, wieso die direkte Anwendung der submodularen Funktion auf die hoch dimensionalen Daten scheitert. Danach werden im Kapitel 4 und 5 die Verfahren Deep Embedded Clustering und Deep Submodular Autoencoder vorgestellt. Insbesondere setzt sich das Kapitel 5 mit den möglichen Problemen des Deep Embedded Clusterings auseinander und wie diese Probleme anhand des Deep Submodular Autoencoders behoben werden können. Anschließend folgt im Kapitel 6 die praktische Umsetzung der vorgestellten Verfahren. Das Ziel ist die Erstellung einer guten Zusammenfassung für die hoch-dimensionalen Datensätze. Vor allem wird im Abschnitt 6.3 das Deep Embedded Clustering genauer auf die möglichen Probleme untersucht, die im Kapitel 5 eingeführt wurden. Im Abschnitt 6.4 wird eine Implementation des Deep Submodular Autoencoders vorgestellt und auf die Herausforderungen, die bei der Implementierung vorkommenden, näher eingegangen. Das Kapitel 7 schließt die Arbeit mit einer Zusammenfassung der wichtigsten Ergebnissen und Bemerkungen ab.

Kapitel 2

Submodulare Funktionen

Für die mathematische Darstellung einer Zusammenfassung eignet sich die Einführung der Mengenfunktion. Eine Mengenfunktion $f : 2^V \to \mathbb{R}$ ist eine Funktion, die jeder Teilmenge einer endlichen Grundmenge V (engl. *Ground set*) eine reelle Maßzahl $f(S)$ (engl. *Score*) zuweist. In Bezug auf die Abbildung 2.1 stellt die Anzahl der unterschiedlichen Farben der Kugeln in einer Urne eine Mengenfunktion dar. Hierbei beschreibt die Grundmenge die Sammlung aller vorhandenen Kugeln. Vor dem Einfügen der blauen Kugel befinden sich in der linken Urne 5 Kugeln und in der rechten Urne 7 Kugeln. Die Mengenfunktion bildet die Menge der Kugeln in der linken Urne auf einen Score von 2 ab, da nur zwei verschiedene Farben (Rot und Grün) zu sehen sind. Hingegen bekommt die rechte Urne einen Score von 3, da drei unterschiedliche Farben (Rot, Grün und Blau) zu erkennen sind. Bezogen auf die Erstellung einer Zusammenfassung ist eine kompakte, repräsentative Teilmenge der Kugeln in der Urne zu suchen. Die Definition einer *repräsentativen Teilmenge* sorgt aber für Unklarheiten. Dem zufolge muss zunächst die Frage gestellt werden, unter welchem Merkmal die Daten zusammengefasst werden sollen. In der Abbildung 2.1 liegt das Interesse beispielsweise auf den Farben der Kugeln in der Urne. Die repräsentativste Zusammenfassung enthält also alle Farben, die in einer Urne vorkommen. Für die Kompaktheit der Zusammenfassung genügt es jeweils einen Repräsentanten für jede Farbe auszuwählen. Folglich repräsentieren eine grüne und eine braune Kugel die optimale Zusammenfassung für die linke Urne, während eine blaue, eine grüne und eine braune Kugel die optimale Zusammenfassung für die rechte Urne darstellen. Um die Repräsentativität einer Menge zu maximieren, muss der Scores einer gewählten Mengenfunktion ebenfalls maximiert werden.

Unter allen Klassen der Mengenfunktionen zeichnen sich die submodularen Funktionen für ihre praktische Anwendungen besonders aus. Der Grund liegt an der besonderen Eigenschaft der submodularen Funktion bei der Erstellung einer Zusammenfassung: Je größer die Zusammenfassung wird, desto wertloser werden die Daten, die nicht in der Zusammenfassung enthalten sind und je kleiner die Zusammenfassung wird, desto wertvoller

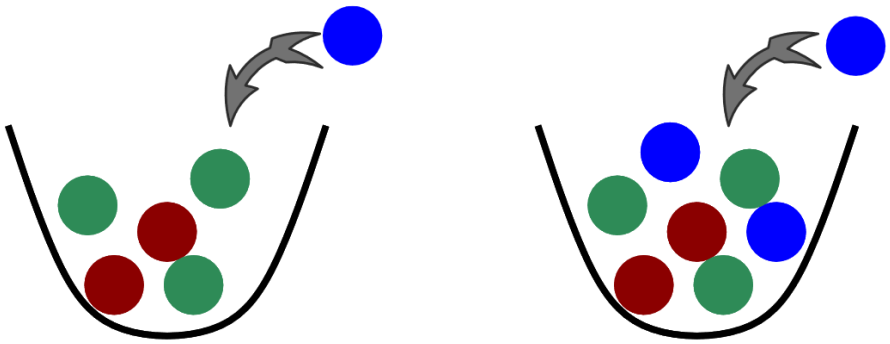


Abbildung 2.1: Illustrierung der Eigenschaft des sinkenden Ertragszuwachses (egl. *Diminishing Return Property*) der Submodularität. Der Zuwachs am Score (*Marginaler Zuwachs*) beträgt 1 bei der linken Urne und 0 bei der rechten Urne durch das Hinzufügen einer neuen blauen Kugel. [9]

werden die nicht in der Zusammenfassung enthaltenen Daten (*Eigenschaft des sinkenden Ertragszuwachses*). Die Abbildung 2.1 illustriert diese Eigenschaft für die beiden Zusammenfassungen, wobei die Zusammenfassung der linken Urne eine Teilmenge der Zusammenfassung der rechten Urne entspricht. Bei beiden Zusammenfassungen wird ein neuer, blauer Kugel hinzugefügt. Der Score inkrementiert sich für die Zusammenfassung der linken Urne auf eins. Für die Zusammenfassung der rechten Urne hat die neue, blaue Kugel keinen Einfluss auf die Repräsentativität, da der Score unverändert bleibt. Folglich ist die neue Kugel wertvoller für die Zusammenfassung der linken Urne als für die Zusammenfassung der rechten Urne.

In diesem Kapitel wird das Konzept der Submodularität und deren Eigenschaft zur Erstellung einer bedeutungsvollen Datenzusammenfassung näher vorgestellt. Im Abschnitt 2.1 werden wichtige Grundbegriffe in Bezug auf die Submodularität präsentiert. Anschließend sollen im Abschnitt 2.2 konkrete Ziele sowie einige bekannte Beispiele der submodulare Funktionen vorgestellt werden. Insbesondere wird im Abschnitt 2.2.1 die Zielfunktion des Informative Vector Machine, sowohl im Kontext des Gaußprozesses als auch in der Beziehung zur Submodularität, kurz präsentiert. Anschließend werden im Abschnitt 2.3 das Optimierungsproblem und die zugehörigen Verfahren vorgestellt, um eine Zusammenfassung mit dem höchsten Score zu konstruieren.

2.1 Grundbegriffe

Die submodulare Funktion ist eine Klasse der Mengenfunktion mit der Eigenschaft der Submodularität. Dabei lässt sich die Submodularität anhand des marginalen Zuwachses (engl. *marginal gain*), auch Ertrag (engl. *discrete derivative*) genannt, definieren.

2.1.1 Definition (Marginaler Zuwachs). Gegeben ist eine Mengenfunktion $f : 2^V \rightarrow \mathbb{R}$, eine Teilmenge $S \subseteq V$ und ein Element $e \in V$. Dann lässt sich der marginale Zuwachs von f für die Teilmenge S in Bezug auf das Element x wie folgt darstellen [32]:

$$\Delta_f(x|S) := f(S \cup \{x\}) - f(S) \tag{2.1}$$

Hierbei beschreibt der marginale Zuwachs die Zunahme des Scores $f(S)$ durch die Aufnahme des Elements x zu der Menge S. Mit der Definition des marginalen Zuwachses lässt sich die Submodularität nun wie folgt definieren.

2.1.2 Definition (Submodularität). Eine Funktion $f : 2^V \rightarrow \mathbb{R}$ ist submodular, wenn für alle $A \subseteq B \subseteq V$ und $x \in V \backslash B$ gilt [32]:

$$\Delta_f(x|A) \geq \Delta_f(x|B) \tag{2.2}$$

Die obige Formel beschreibt die Eigenschaft des sinkenden Ertragszuwachses (engl. *Diminishing Return Property*). Der Informationsgewinn (Score) durch die Aufnahme eines neuen Elements in eine kleinere Menge ist größer als der Informationsgewinn durch die Aufnahme desselben Elements in eine größere Menge. Der Ertrag reduziert sich also bei der Aufnahme eines neuen Elements in einem größeren Zusammenhang [8]. Wegen dieser Eigenschaft ist die Verwendung der submodularen Funktion ideal für die Erstellung einer optimalen Datenzusammenfassung.
Weitere wichtige Eigenschaften der submodularen Mengenfunktion sind die Normalisierung, die Nicht-Negativität, und die Monotonie.

2.1.3 Definition (Normalisierung). Eine submodulare Funktion $f : 2^V \rightarrow \mathbb{R}$ ist zudem normalisiert, falls gilt [8]:

$$f(\emptyset) = 0 \tag{2.3}$$

2.1.4 Definition (Nicht-Negativität). Eine submodulare Funktion $f : 2^V \rightarrow \mathbb{R}$ ist dazu nicht-negativ, falls für alle $S \subseteq V$ gilt [8]:

$$f(S) \geq 0 \tag{2.4}$$

2.1.5 Definition (Monotonie). Zusätzlich ist die Funktion f monoton, wenn für alle $x \in V$ und für alle $S \subseteq V$ gilt [32]:

$$\Delta_f(x|S) \geq 0 \tag{2.5}$$

Die Monotonieeigenschaft besagt, dass der Score durch die Aufnahme eines beliebigen Elements in eine Teilmenge S nie kleiner werden kann. Da sie eine ertragbare Annahme ist und das Umgehen mit dem Optimierungsproblem vereinfacht, wird sie oft bei der Auswahl einer submodularen Funktion berücksichtigt.

2.2 Submodulare Funktionen

Für das Datenzusammenfassungsproblem wird nun eine nicht-negative, normalisierte, submodulare Zielfunktion f ausgesucht. In [41] wurde eine allgemeine Vorgehensweise für die Erstellung solcher Zielfunktionen präsentiert. Sie besagt, die Relevanz und die Nicht-Redundanz seien die zwei wichtigsten Eigenschaften für eine gute Zusammenfassung. Dabei messen die Zielfunktionen für Zusammenfassungen diese beiden Eigenschaften getrennt. Die Relevanz deutet darauf hin, wie gut die Zusammenfassung die Grundmenge repräsentiert und die Nicht-Redundanz darauf, wie sehr die einzelnen Beobachtungen in der Zusammenfassung sich voneinander unterscheiden. Deswegen soll in der Funktion die Relevanz gefördert und die Redundanz der Daten bestraft werden: $f(S) = L(S) + \lambda R(S)$, wobei $L(S)$ die Vollständigkeit (egl. *Coverage*) der Information von der Zusammenfassung S in Bezug auf die Grundmenge misst, $R(S)$ die Diversität innerhalb von S belohnt und $\lambda \geq 0$ einen Regulierungsparameter darstellt. Da die Bestrafung der Redundanz gegen die Monotonie der Zielfunktion spricht [12] [41], wird die Diversität der Zusammenfassung gefördert, statt die Redundanz zu bestrafen. Die klassischen Beispiele der submodularen Funktionen sind die gewichtete Coverage-Funktion (*weighted coverage function*) mit einer monotonen Gewichtsfunktion [32] und die Maximierung der positiven Services/Einträgen bezüglich des Versorgungsstandorte-Problems (*Facility location*) [22]. Des Weiteren kann die Verlustfunktion des k-Medoids Problems [31] anwendet werden, um eine submodulare Funktion zu erstellen. Hierbei handelt es sich um die Reduzierung des Verlusts durch das Hinzufügen eines Hilfselements in die Zentroidmenge (*Exemplar-based Clustering*) [23]. Die logarithmische Determinante der Kernel-Funktion von der *Informative Vector Machine* stellt ebenfalls eine submodulare Funktion dar [36], welche im folgenden Abschnitt genauer vorgestellt wird.

2.2.1 Informative Vector Machine

Im Bereich der nicht-parametrisierten Regressionsanalyse ist der Gaußprozess zu einem beliebten Approximationsverfahren geworden [51]. Jedoch zeigt das Verfahren seine Schwierigkeit in der praktischen Anwendung, aufgrund des kubischen Wachstums der Rechenkomplexität in Bezug auf die Datenmenge. Aus diesem Grund führen Lawrence et al. [36] die Informative Vector Machine ein, um die hohe Abhängigkeit der Rechenkomplexität zur gegebenen Datenmenge zu beseitigen.
Mit der unbekannten, wahren Funktion $f(x)$ ist das Ziel mit N tatsächlichen (gemessenen) Beobachtungen $S = \{(x_i, y_i)\}_{i=1}^{N}$ eine Funktion $\hat{f}(x)$ zu approximieren [5]. Unter der Betrachtung der kleinen Messfehlern bei den tatsächlichen Beobachtungen werden die Daten y_i als verrauscht angenommen, sodass sie von den tatsächlichen Funktionswerten $f(x_i)$ um ϵ abweichen:

$$y_i = \hat{f}(x_i) = f(x_i) + \epsilon. \tag{2.6}$$

Hierbei entspricht die Abweichung ϵ den unabhängigen und identisch verteilten Fehler, der einer Normalverteilung $\mathcal{N}(0, \sigma^2_{noise})$ folgt [10]. Folglich muss die approximierte Funktion $\hat{f}(x)$ auch die Abweichung bei ihrer Vorhersage in Betracht ziehen. Die Gaußprozess-Regression ist ein Bayes'scher Ansatz, bei dem die a-priori-Verteilung der stetigen Funktionen einen Gaußprozess darstellt [5]:

$$f|X \backsim \mathcal{N}(\mu, \mathrm{K}), \tag{2.7}$$

wobei $f = (f(x_1), f(x_2), \ldots, f(x_N))$ einen Vektor der Ausgaben, $X = (x_1, x_2, \ldots, x_N)$ eine Matrix der Eingaben (mit $x_i \in \mathbb{R}^d$), $\mu = (\mu(x_1), \mu(x_2), \ldots, \mu(x_N))$ einen Erwartungsvektor der Mittelwertfunktion $\mu(x)$ und $\mathrm{K} = [\kappa(x_i, x_j)]^N_{i,j=1}$ eine Kovarianzmatrix darstellt. Der Gaußprozess modelliert also eine Normalverteilung auf Funktionen in Bezug auf den vorgegebenen Datensatz X. Offen steht noch die Auswahl der Kovarianzfunktion κ. Als eine geeignete Kovarianzfunktion kann die Kernelfunktion eingesetzt werden [36]. In einem Datenraum χ stellt eine Kernel-Funktion $\kappa(x_i, x_j)$ mit $\kappa : \chi \times \chi \to \mathbb{R}_{\geq 0}$ die Ähnlichkeit zweier Beobachtungen x_i, x_j dar. Somit stellt sich die Funktion als ein möglicher Ansatz, um die Diversität der Beobachtungen zu berechnen. Ein bekanntes Beispiel für eine Kernel-Funktion ist der *RBF Kernel*:

$$\kappa(x_i, x_j) = \exp\left(-\frac{1}{2l^2} \cdot ||x_i - x_j||_2^2\right) \tag{2.8}$$

wobei $l \in \mathbb{R}$ eine skalierbare Konstante und $||\cdot||_2$ die euklidische Norm darstellen. Anhand der Kernel-Funktion lässt sich die Kernel-Matrix $\mathrm{K} = [\kappa(x_i, x_j)]^N_{i,j=1}$ als die Kovarianzmatrix des Gaußprozesse 2.7 aufstellen. Sie enthält die Ähnlichkeit aller Paare der Daten in X, sodass Paare, die ähnlicher sind, einem höheren Wert zugewiesen werden. Auf der Hauptdiagonale der Kernelmatrix liegen die größten Werte, da die Elemente zu sich selbst am ähnlichsten sind. Da aber das Interesse auf die Diversität der Paare liegt, werden Paare betrachtet, die einen Ähnlichkeitswert nahe 0 aufweisen [11]. Aus der Summe der beiden unabhängigen, normalverteilten Zufallsvariablen aus 2.6, 2.7 ergibt sich ein Gaußprozess mit [3]:

$$\hat{f}(x_i)|X \backsim \mathcal{N}(0, \mathrm{K}(X, X) + \sigma^2_{noise}) \tag{2.9}$$

wobei $\hat{f}(x_i)|X$ die a-priori-Verteilung der approximierten Funktionen darstellt. Aus der multivariaten Verteilung zwischen der Verteilung aus 2.9 und der Verteilung aus 2.7 lässt sich die folgende bedingte Wahrscheinlichkeitsverteilung für die Vorhersage aufstellen:

$$P(y_*|X_*, X, y) = \mathcal{N}(\mu_*, \Sigma_*) \tag{2.10}$$

mit $\mu_* = \mathrm{K}(X_*, X)[\mathrm{K}(X, X) + \sigma^2_{noice} I]^{-1} y$ und $\Sigma_* = \mathrm{K}(X_*, X_*) - \mathrm{K}(X_*, X)[\mathrm{K}(X, X) + \sigma^2_{noise} I]^{-1}\mathrm{K}(X, X_*)$, wobei I die Identitätsmatrix entspricht [3] [10].

Wie oben angesprochen ist die größte Schwäche der Gaußprozess-Regression die Abhängigkeit der Rechenkomplexität von der Anzahl der Beobachtungen. Aufgrund der Invertierung der Kernel-Matrix in 2.10 benötigt das Verfahren eine kubische Laufzeit $O(|X|^3)$

und zusätzlich eine Speicherkapazität von $O(|X|^2)$. Das Ziel ist die Anzahl der Beobachtungen für den Gaußprozess zu reduzieren. Problematisch ist jedoch, dass die Qualität der Regressionsanalyse stark von der Anzahl der tatsächlichen Beobachtungen abhängt. Bezogen auf den Gaußprozess 2.7 führt die Reduzierung der Datenmenge zu einer hohen Flexibilität der möglichen Funktionen aus der Normalverteilung und schließlich zu einer hohen Unsicherheit der Regression. Dementsprechend versucht die Informative Vector Machine eine Teilmenge aus $k << |X|$ ausschlaggebenden Beobachtungen zu finden, welche ein ausreichend gutes Ergebnis liefert. Wenn die ausschlaggebenden Beobachtungen als die Repräsentanten der gesamten Daten betrachtet werden und die Repräsentativität auf die „Sicherheit“ des Gaußprozesses basiert, dann ähnelt das folgende Problem an das vorgestellte Datenzusammenfassungsproblem. Um die Sicherheit einer Verteilung zu messen, kann die Entropie verwendet werden [36]. Eine Entropie ist ein nicht-negatives Maß für den Grad der Unsicherheit einer Verteilung [51]. Das Minimierungsproblem[1] lässt sich leicht zu einem Maximierungsproblem umwandeln, indem der Fokus auf die Differenz der Entropien vor und nach dem Hinzufügen einer neuen Beobachtung in die Teilmenge verlegt wird. Anhand eines gierigen Ansatzes (Kapitel 2.3.1), wählt die Informative Vector Machine Beobachtungen in die Teilmenge, die die größte Entropieabnahme erzeugt [35], bis die Teilmenge k Beobachtungen enthält. Dieser Ansatz lässt sich als eine Maximierung der logarithmischen Determinante der Kernel-Funktion aufschreiben:

$$f(S) = \frac{1}{2} \log \det(\mathcal{I} + a\mathrm{K}(S, S)) \tag{2.11}$$

wobei S die aktuelle Teilmenge, $a \in \mathbb{R}_+$ ein skalierbarer Parameter und $\mathcal{I}$ eine Einheitsmatrix darstellt. In [56] wurde bewiesen, dass die Funktion submodular und monoton ist. Das bedeutet, dass für eine leere Menge die Entropie am höchsten ist und das Hinzufügen einer neuen Beobachtung in eine kleinere Menge immer zu einer höheren Entropieabnahme führt. Außerdem ist die Funktion zusätzlich normiert, da für eine leere Menge die Kernel-Matrix leer ist und aus dem Logarithmus der Determinante von der Einheitsmatrix in den Wert 0 resultiert. Insgesamt schafft die Informative Vector Machine eine Laufzeit von $O(k^2 \cdot |X|)$ mit einer Speicherkapazität von $O(k \cdot |X|)$ zu erreichen [35].

[1] Minimierung der Entropie

2.3 Submodulare Maximierung

Das Ziel der submodularen Maximierung ist eine optimale Zusammenfassung zu finden, welche eine gegebene, nicht-negative, monotone, submodulare Mengenfunktion $f : S^V \to \mathbb{R}$ mit $f(\emptyset) = 0$ maximiert [11]. Formal lässt es sich als das folgende Maximierungsproblem darstellen [32]:

$$\arg\max_{S \subseteq V} f(S) \text{ unter Berücksichtigung der Nebenbedingung für S} \tag{2.12}$$

Hierbei beschreibt das Maximierungsproblem unter welcher Nebenbedingung die Zusammenfassung stattfinden sollen. Die Kardinalitätsbeschränkung ist eine mögliche Nebenbedingung, welche besagt, dass die suchende Teilmenge höchstens k Elemente, wobei $k \in \mathbb{N}_{\geq 0}$, besitzen darf. Formal ist das Maximierungsproblem unter der Kardinalitätsbeschränkung definiert als:

$$f^* := \arg\max_{S \subseteq V, |S| \leq k} f(S) \tag{2.13}$$

Das folgende Problem aus 2.13 ist jedoch für viele Klassen der submodularen Funktionen NP-Schwer [32]. Folglich verlegt sich der Fokus auf die Erstellung eines approximativen Algorithmus. Die Optimalität eines approximativen Algorithmus wird anhand seiner Approximationsgarantie α bewertet, wobei $0 \leq \alpha \leq 1$ gilt. Bezogen auf die submodulare Maximierung wird nach einer Menge $S_{approx} \subseteq V$ gesucht, welche eine möglichst hohe Approximationsgarantie α erzeugt und die folgende Gleichung erfüllt:

$$OPT \geq f(S_{approx}) \geq \alpha OPT \tag{2.14}$$

wobei $OPT = f(S^*)$ den Score der optimalen Zusammenfassung S^* und $f(S_{approx})$ den Score der berechneten Zusammenfassung S_{approx} darstellen. Die Approximationsgarantie α beschreibt somit, um mindestens wie viel Faktor der Score $f(S_{approx})$ den optimalen Score OPT erreicht. Es existieren bereits viele Verfahren, die eine angenehme Approximationsgarantie liefern. In diesem Kapitel wird ein gierig-basiertes Verfahren präsentiert, welche eine zufriedenstellende Approximationsgarantie in linearer Zeit liefert. Weitere approximative Ansätze zur Berechnung des submodularen Maximierungsproblems befinden sich im Anhang.

2.3.1 Gieriger Algorithmus

Eine einfache und bekannte Technik für Approximationsalgorithmen ist die Verwendung der gierigen Methode. Die Grundidee ist der iterative Aufbau einer Lösung nach einem lokalen Optimum. Nemhauser et al. [50] präsentieren im Jahr 1978 den gierigen Algorithmus in Bezug auf das Maximierungsproblem 2.13.

Algorithmus 1 Der gierige Algorithmus von Nemhauser et al [50]

Eingabe: Grundmenge V, Kardinalität M, Submodulare Funktion f
Ausgabe: Zusammenfassung S_{greedy}
  $S_{greedy} \leftarrow \emptyset$
  for $1, \ldots, M$ **do**
    $x = \arg\max\{\Delta_f(x|S_{greedy})|x \in V\}$
    $S_{greedy} \leftarrow S_{greedy} \cup \{x\}$
  end for
  return S_{greedy}

Initialisiert wird zuerst eine leere Menge für die Zusammenfassung. Bei jeder Iteration fügt der Algorithmus das Element, welches den größten marginalen Zuwachs erzeugt, in die Zusammenfassung ein. In [50] wurde bewiesen, dass für eine nicht-negative, monotone, submodulare Funktion: $f(S_{greedy}) \geq (1 - (1/\exp(1)))\, f(S^*)$ gilt, wobei S_{greedy} die gierige Zusammenfassung und S^* die optimale Zusammenfassung darstellt. Folglich ergibt sich aus dem gierigen Ansatz eine Approximationsgarantie von mindestens $(1 - (1/\exp(1))) \approx 63\%$. Insgesamt bietet der grierige Algorithmus eine Speicherkapazität von $O(k)$ und eine Laufzeit von $O(k \cdot |V|)$ Funktionsauswertungen [2].

Kapitel 3

Autoencoder

Die submodularen Funktionen bieten eine gute Grundlage für eine optimale Zusammenfassung. Die Einführung des Maximierungsproblems (Gleichung 2.12) im Kapitel 2.3 und die vorgestellten Approxiationsalgorithmen in den Kapiteln 2.3.1 und A.0.1 ermöglichen gleichzeitig eine aussagekräftige Zusammenfassung zu erstellen. Darunter zeichnet sich die log-Determinante-Funktion (Gleichung 2.11) als eine gute Wahl aus, da sie ihren Fokus auf die Maximierung der Diversität der ausgewählten Teilmenge legt. Problematisch wird die Verwendung von Kernel-Funktionen, sobald unstrukturierte, hoch-dimensionale Rohdaten betrachtet werden, wie beispielsweise Bilder oder Zeitreihen [53]. In diesem Kapitel wird zuerst näher erläutert, wieso Kernel-Funktionen, insbesondere der RBF-Kernel, ihre Schwierigkeiten bei zunehmender Dimensionalität zeigen. Vor diesem Hintergrund wird folglich der Autoencoder eingeführt, welches das Problem der Hoch-Dimensionalität mit einem unüberwachten, Deep-Learning basierten Ansatz löst. Ein Autoencoder ist ein spezielles, neuronales Netz, welches eine Abbildung von unstrukturierten, hoch-dimensionalen Daten auf einen niedrig-dimensionalen Raum lernt. Darunter zeichnet sich der Autoencoder als ein beliebter Deep-Learning-Ansatz für eine effektive Dimensionsreduktion aus. Schließlich werden verschiedene Erweiterungen von Autoencodern vorgestellt, die einerseits die Abbildung verfeinern, andererseits nützlich für andere Anwendungen, wie beispielsweise Merkmalsrepräsentation [6] oder Anomalieerkennung [62] [54] sein können.

3.1 Fluch der Dimensionalität

Der Fluch der Dimensionalität ist ein häufig vorkommendes Phänomen im Bereich des maschinellen Lernens und Data Minings. Der Begriff wurde von Richard E. Bellman im Jahr 1961 eingeführt [4] und beschreibt die Schwierigkeit vieler maschinellen Lernmethoden in Bezug auf die steigende Anzahl der Dimensionalität. Insbesondere betroffen sind auch die Distanz-Metriken. Beyer et. al. haben das Verhalten der Distanz-Metriken in der hohen Dimension untersucht [7]. Dabei erfolgt die Untersuchung anhand des Nächsten-

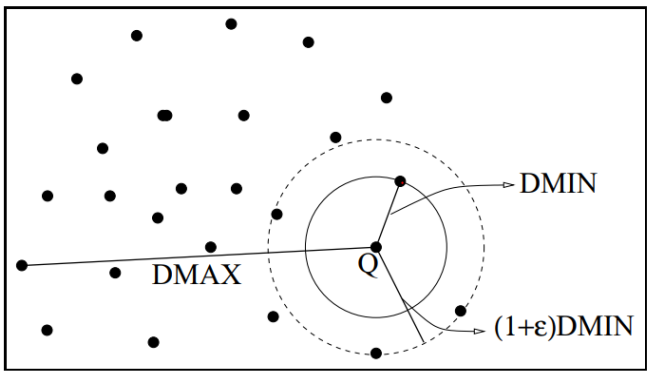


Abbildung 3.1: Illustrierung des Nächsten-Nachbar-Problems im Bezug auf den Zentroid Q (Bezugspunkt egl. *Query Point*). DMIN stellt die Distanz zum nächsten Nachbar und DMAX stellt die maximale Distanz dar. Der Kreis mit dem Radius DMIN bestimmt die Zentroid-Region, wobei der Kreis mit dem Radius $(1 + \varepsilon)$DMIN sich auf die erweiterte Zentroid-Region bezieht [7].

Nachbar-Problems (engl. Nearest Neighbor Problem). Es handelt sich um die Bestimmung des Punktes in einem Datensatz, welcher einem gegebenen Zentroid-Punkt z_i (Bezugspunkt egl. *Query Point*) am nächsten liegt. Eingeführt wird zunächst der Begriff der Instabilität des Nächsten-Nachbar-Zentroids. Ein Zentroid sei für ein gegebenes ε instabil, falls die Distanz des Zentroids zu den meisten Datenpunkten kleiner als das $(1 + \varepsilon)$-fache der Distanz des Zentroids zu seinem nächsten Nachbarn ist [7]. Bezogen auf die Abbildung 3.1 bedeutet die Definition für ein kleines ε: je mehr Punkte sich in der erweiterten Zentroid-Region befinden, desto schwieriger wird die Unterscheidung des nächsten Nachbarn von den restlichen Punkten. Um zu zeigen, dass die Wahrscheinlichkeit der Instabilität für alle $\varepsilon > 0$ mit zunehmender Dimensionalität gegen 1 konvergiert, wird das folgende Theorem aufgestellt.

3.1.1 Definition. Für die Darstellung des Theorems sind folgende Definitionen und Notationen zunächst notwendig [7]:

- Seien $\forall d\ X_{d,1}, \ldots, X_{d,n}$ n unabhängige Datenpunkte, die einer Daten-Verteilung aus der Dimension d folgen.
- Sei Z_d ein Zentroid-Punkt, welcher einer Zentroid-Verteilungen aus der Dimension d folgt und unabhängig von allen $X_{d,i}$ für $1 \leq i \leq n$ gewählt wurde.
- $\text{DMIN}_d = \min \{||X_{d,i} - Z_d|| \text{ für } 1 \leq i \leq n\}$
- $\text{DMAX}_d = \max \{||X_{d,i} - Z_d|| \text{ für } 1 \leq i \leq n\}$

3.1.2 Theorem. *Anhand der Definition 3.1.1 lässt sich das Theorem von Beyer et al. darstellen [7].*

$$Sei\ lim_{d\to\infty}\ var(\frac{||X_{d,1} - Z_d||_k}{E[||X_{d,1} - Z_d||_k]}) = 0,$$

dann folgt für alle $\varepsilon > 0$*:*

$$lim_{d\to\infty}\ P[DMAX_d \leq (1+\varepsilon)DMIN_d] = 1$$

wobei $var(\cdot)$ *die Varianz,* $E[\cdot]$ *den Erwartungswert,* $P[\cdot]$ *die Wahrscheinlichkeit und* $||\cdot||_k$ *die gewählte Distanzmetrik darstellt.*

Das Theorem zeigt, wie sich die Distanz-Verteilung mit der wachsenden Dimensionalität d verhält. Es besagt: unter einer relativ allgemeinen Voraussetzung der Distanz-Verteilung (mehr dazu in [7]) konvergieren sich die maximale und die minimale Distanz. Das heißt, alle Punkte konvergieren mit der wachsenden Dimensionalität gegen eine gleiche Distanz vom Zentroid. Dieses Theorem deutet auf die Instabilität des Nächsten-Nachbar-Zentroids hin, aufgrund des Verhaltens der Distanzmetriken unter einer zunehmenden Dimensionalität. Das Verhalten der Distanzmetriken wurde in [1] näher beschrieben und weitergeführt. Dabei zeigen Aggarwal et al., dass die Differenz $DMAX_d - DMIN_d$ bei der zunehmenden Dimensionalität unabhängig von der Daten-Verteilung mit einer Rate von $d^{1/k-1/2}$ steigt, wobei k sich auf die gewählte Distanzmetrik $||\cdot||_k$ bezieht. In der Abbildung 3.2 wird die Überlegung auf die verschiedenen Distanzmetriken durchgeführt. Festgestellt wurden die verschiedenen Verhaltensweisen der Metriken. Bei der Manhattan-Distanz $k = 1$ divergiert die Differenz auf ∞, bei der Euklidische-Distanz $k = 2$ konvergiert die Differenz gegen eine Konstante und für $k \geq 2$ konvergiert die Differenz gegen 0.

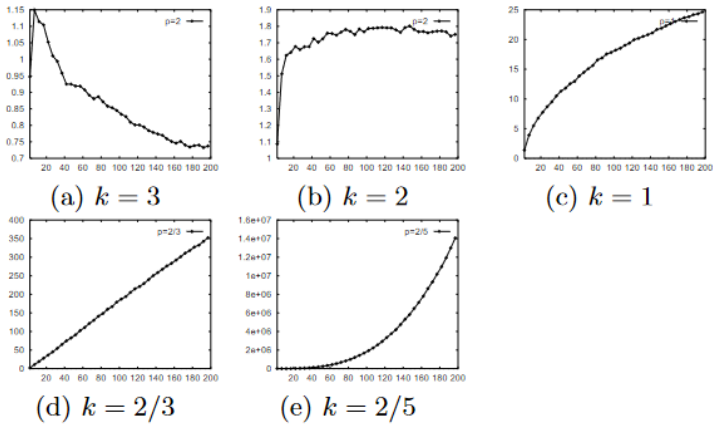


Abbildung 3.2: Darstellung von dem Differenz $DMAX_d - DMIN_d$ im Bezug auf die Dimensionalität, wobei k dem Parameter für die gewählte Distanzmetrik $||\cdot||_k$ entspricht (uniform data) [1].

Die Unsicherheit durch die steigende Dimensionalität tretet ebenfalls bei der Kernel-basierten, submodularen Funktionen auf. Vor allem verwendet der RBF-Kernel die Euklidische-Distanz, sodass er seine Aussagekraft in der hohen Dimension verliert. Die niedrige Aussagekraft führt schließlich zum Verlust der Vertrauenswürdigkeit der erstellten Datenzusammenfassung.

3.2 Tiefes Neuronales Netzwerk

Die Kernel-Funktionen versprechen zwar eine gewisse Interpretierbarkeit in der niedrigen Dimension, jedoch scheitern die Funktionen daran, ihre Aufgabe in der hohen Dimension zu erfüllen. Diese starke Abhängigkeit zwischen dem Kernel und der Dimensionalität führt zur Notwendigkeit der Datenvorverarbeitung. Gewünscht ist, unstrukturierte, hochdimensionale Daten auf einen niedrig-dimensionalen, eingebetteten Raum abzubilden. Ein wichtiger Aspekt bei der Abbildung ist die Einhaltung des Distanzverhältnisses der höheren Dimension. Ein (tiefes) neuronale Netz, ein Deep-Learning basierter Ansatz, ist ein nicht-lineares Verfahren, um eine erfolgreiche Abbildung vom hoch-dimensionalen Raum auf den niedrig-dimensionalen, repräsentativen Raum zu erstellen [18].

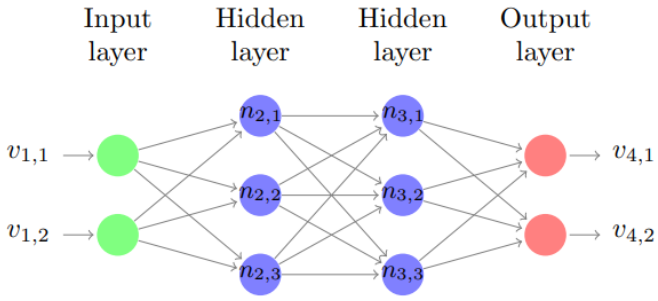
3.2.1 Grundlage

Ein (tiefes) neuronales Netzwerk (engl. *Deep Neural Network*) , DNN $\mathcal{N}$, besteht aus mehreren Neuronen (Knoten) $n_{k,l}$ mit $2 \leq k \leq K$, $1 \leq l \leq s_k$ und Schichten $\{\mathbb{S}_1..\mathbb{S}_K\}$, wobei K die Anzahl der Schichten darstellt [30]. Jede Schicht $\mathbb{S}_k$ enthält s_k Neuronen. Formal lässt sich ein DNN als das folgende 3-Tupel definieren:

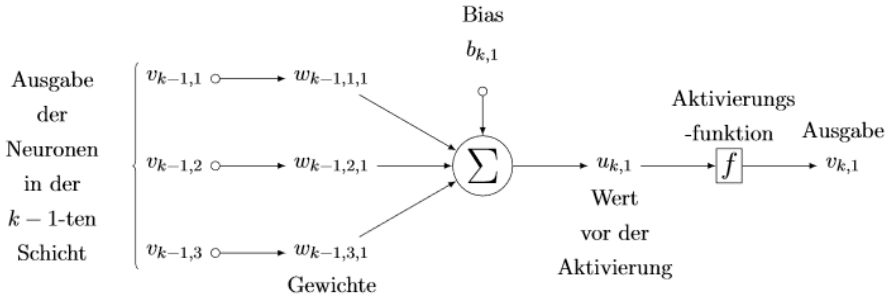
$$\mathcal{N} = (\mathbb{S}, \mathbb{T}, \Phi),$$

wobei $\mathbb{S} = \{\mathbb{S}_k | k \in \{1..K\}\}$ die Menge der Schichten, $\mathbb{T} \subseteq \mathbb{S} \times \mathbb{S}$ die Menge der Verbindungen zwischen den Schichten und $\Phi = \{\phi_k | k \in \{2..K\}\}$ die Menge der Funktionen für die Neuronen darstellt [30]. Die Menge der Neuronen kann zwischen den Eingabe-Neuronen $n_{1,l}$ in der Eingabe-Schicht (egl. *Input layer*) $\mathbb{S}_1$, den Ausgabe-Neuronen $n_{K,l}$ in der Ausgabe-Schicht (egl. *Output layer*) $\mathbb{S}_K$ und den versteckten Neuronen, die in den versteckten-Schichten (egl. *Hidden layer*) $\mathbb{S}_k$ mit $1 < k < K$ liegen, unterschieden werden (Abbildung 3.3 a) [43]. Diese Neuronen der unterschiedlichen Schichten sind mit gewichteten Kanten verbunden, welche den Transport der Informationen von der Ausgabe eines Neurons zur Eingabe eines nächsten Neurons ermöglichen. Mit anderen Worten ist das Neuron $n_{k-1,l'}$ von der $k-1$-ten Schicht mit dem Neuron $n_{k,l''}$ von der $k$ten Schicht mit einem Gewicht von $w_{k-1,l',l''}$ verbunden.

Des Weiteren werden zwei Variablen $u_{k,l}$,$v_{k,l}$ jedem Neuron $n_{k,l}$ zugeordnet. Sie stellen Werte vor und nach der Aktivierungsfunktion dar [30]. Der Eingabewert $u_{k,l}$ der Aktivie-



(a) Illustrierung eines tiefen, neuronalen Netzes [30]



(b) Darstellung der Funktionsweise des Neurons $n_{k,1}$

Abbildung 3.3: Ein tiefes, neuronales Netzwerk

rungsfunktion lässt sich mit der linearen Kombination aus den Ausgabewerten der verbundenen Neuronen in der vorherigen Schicht $v_{k-1,l'}$, den Kantengewichten $w_{k-1,l',l}$ und dem Bias b berechnen [30]. Aktivierungsfunktionen dienen zur Bestimmung der Ausgabe $v_{k,l}$ der Neuronen. Bekannte Aktivierungsfunktionen sind Rectified Linear Unit ($ReLU$) und Sigmoid Funktion. $ReLu$, $v_{k,l} = max(0, u_{k,l})$, wandelt die Eingabe $u_{k,l}$ in ein Maximum zwischen $u_{k,l}$ und 0 um [30]. Somit führen die negativen Eingaben zur Deaktivierung der Neuronen und werden nicht weitergereicht. Die Sigmoid Funktion, $v_{k,l} = \frac{1}{1+e^{-u_{k,l}}}$, normalisiert die Eingaben im Bereich zwischen 0 und 1. Je größer die Eingabe $u_{k,l}$ desto näher ist der Ausgabewert $v_{k,l}$ an 1 und je negativer die Eingabe $u_{k,l}$ desto näher ist die Ausgabe $v_{k,l}$ an 0 [59]. Die Abbildung 3.3 b präsentiert die vorgestellte Funktionsweise des Neurons $n_{k,l}$ mit $l = 1$ in der k-ten Schicht. Die Werte $v_{k-1,1}, v_{k-1,2}, v_{k-1,1}$ stellen die Ausgabewerte der Neuronen in der vorherigen Schicht $k-1$ dar. Folglich repräsentieren die Werte $w_{k-1,1,1}, w_{k-1,2,1}, w_{k-1,3,1}$ die Gewichte der Kanten, die von den Neuronen $n_{k-1,1}, n_{k-1,2}, n_{k-1,3}$ aus der $k-1$-ten Schicht zu dem betrachteten Neuron $n_{k,1}$ verbunden sind. Der Wert $u_{k,1}$ ergibt sich aus der Berechnung $b_{k,1} + \Sigma_{i=1}^{3} v_{k-1,i} w_{k-1,i,1}$ und der Wert $v_{k,1}$ aus der Aktivierungsfunktion $f(u_{k,1})$. Der gesamte Prozess der Informationsweitergabe in der Vorwärtsrichtung durch das Netz wird als *Forward Propagation* genannt [17].

3.2.2 Gradientenverfahren

Das Ziel der Trainingsphase (Lernphase) ist, die Parameter des Netzes (beispielsweise die Gewichte) so zu optimieren, sodass der Vorhersagefehler minimiert wird. Der Vorhersagefehler, auch Verlust genannt, dient zur Evaluierung des vorhergesagten Ergebnisses und stellt somit den Fehler zwischen dem erwarteten Ergebnis und dem vorhergesagten Ergebnis des Netzes dar. Berechnet wird der Fehler anhand einer ausgewählten Verlustfunktion. Das Problem lässt sich als das folgende Minimierungsproblem formalisieren:

$$\theta^* = \arg\min_{\theta} \; L(x, x') \tag{3.1}$$

wobei $L : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ die Verlustfunktion, x das erwartete Ergebnis des trainierten, neuronalen Netzes, x' das vorhergesagte, tatsächliche Ergebnis und θ die Parameter des Netzes darstellt. Das folgende Minimierungsproblem kann anhand des Gradientenverfahrens (egl. *Gradient Descent Algorithm*) gelöst werden. Vorgestellt wurde das Konzept im Jahr 1847 von Cauchy [13]. Das Gradientenverfahren ist ein Optimierungsverfahren, welches nach einem lokalen Minimum der Verlustfunktion sucht. Berechnet wird zunächst der Gradient der Verlustfunktion im Bezug auf die Parameter θ ($\partial L/\partial\theta$). Schließlich werden die Parameter θ durch mehrere Schritte in der Richtung des steilsten Abstiegs der Verlustfunktion optimiert. Formal lässt sich die Optimierung wie folgt aufstellen:

$$\theta^* = \theta - \gamma \frac{\partial L}{\partial \theta} \tag{3.2}$$

wobei θ^* die optimierten Parameter des Netzes und γ die Lernrate (egl. *Learning rate*), welche die Größe des Optimierungsschritts festlegt, darstellt. Mit der Funktion 3.2 werden die Parameter in der entgegengesetzten Richtung des Gradients der Verlustfunktion mit einem Faktor von γ aktualisiert.

Vorgestellt werden drei Gradientenverfahren. Sie unterscheiden sich jeweils von der Anzahl an Daten, die sie zur Berechnung des Gradients der Verlustfunktion verwenden. Abhängig von der Datenmenge werden Kompromisse zwischen der Genauigkeit der Parameteraktualisierung und der Zeit, die für eine Aktualisierung benötigt wird, eingeführt [52]. Je größer die Datenmenge, die das Verfahren für eine Aktualisierung der Parameter verwendet, desto langsamer ist zwar die Aktualisierung, jedoch kann die Optimierung präziser an die jeweiligen, einzelnen Daten angepasst werden.

Das Batch Gradientenverfahren (BGD) berechnet den Gradient der Verlustfunktion bezüglich der Parameter für die gesamten Daten. Folglich ergibt sich nur eine Aktualisierung (Schritt) der Parameter und das Verfahren bietet eine schnelle Annäherung zum lokalen Minimum. Jedoch kann das BGD durch die Berechnungen mit großen Eingaben sehr langsam sein und der Speicher kann mit einem zu großen Datensatz belastet werden [52].

Das stochastische Gradientenverfahren (SGD) berechnet den Gradient der Verlustfunktion für eine einzelne, zufällige Eingabe von den gesamten Daten. Folglich beschleunigt sich die Berechnung im Vergleich zum BGD [52]. Außerdem führt das SGD zu häufigen Aktualisierungen mit einer hohen Varianz, wodurch starke Schwankungen der Gradienten entstehen. Zwar ermöglicht die starke Schwankung die Suche nach einem potenziell besseren, lokalen Minimum, jedoch kann sie auch die Konvergenz gegen ein exaktes Minimum erschweren [52].

Das (Stochastische) Mini-Batch Gradientenverfahren berechnet den Gradient der Verlustfunktion für eine bestimmte Menge an zufälligen Daten (Mini-Batch). Dadurch garantiert das Verfahren die Schnelligkeit der Gradienten-Berechnungen durch eine kleinere Eingabe als beim BGD und gleichzeitig reduziert sich die Varianz der Aktualisierungen im Vergleich zum SGD, sodass die Konvergenz gegen ein Minimum erleichert wird [52].

Eine komplette Trainingsphase eines Netzes lässt sich zusammengefasst wie folgt darstellen:

1. (Zufällige) Initialisierung der Parameter
2. Führe das Gradientenverfahren durch
   (a) Forward propagation: Berechnung des vorhergesagten Ergebnisses
   (b) Berechnung der Vorhersagefehler anhand einer gewählten Verlustfunktion L
   (c) Berechnung des Gradienten der Verlustfunktion im Bezug auf die Parameter $(\partial L/\partial \delta)$
   (d) Berechnung der neuen Parameter anhand der Funktion 3.2
   (e) Wiederholung des Schritts 2 für weitere Daten, falls das SGD oder das Mini-Batch Gradientenverfahren gewählt wurde

3.2.3 Rückpropagierung

Die Berechnung des Gradients von einem tiefen neuronalen Netz ist jedoch wegen seiner komplexen Struktur und der hohen Parameteranzahl nicht-trivial [17]. Deswegen wird zusätzlich die Rückpropagierung (egl. *Backpropagation*) angewendet, um den Gradient effektiv zu berechnen. Dabei wird zuerst der Gradient der Verlustfunktion in Bezug auf die Parameter zwischen der vorletzten Schicht $\$_{K-1}$ und der letzten Schicht $\$_K$ berechnet. Zur Visualisierung des Prozesses werden die Notationen anhand der Abbildung 3.4 dargestellt.

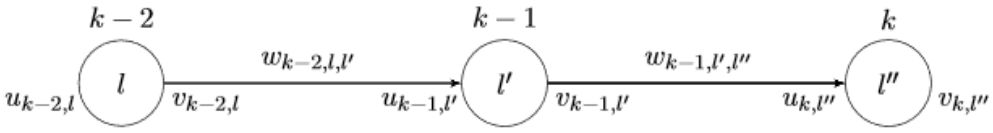


Abbildung 3.4: Darstellung der Notation für die Funktionen 3.3 und 3.4

Hierbei entspricht die k-te Schicht die letzte Schicht $\$_K$ und $k-1$-te Schicht die vorletzte Schicht $\$_{K-1}$. Gesucht ist der Gradient der Verlustfunktion in Bezug auf das Gewicht $w_{k-1,l',l''}$ zwischen zwei Neuronen l' und l'' Des Weiteren entsprechen $u_{k-1,l'}$ und $u_{k,l''}$ die Werte des Neurons l' und l'' vor der Aktivierung und $v_{k-1,l'}$ und $v_{k,l''}$ die Werte des Neurons l' und l'' nach der Aktivierung. Mit Hilfe der Kettenregel kann der Gradient wie folgt berechnet werden:

$$\begin{aligned}
\frac{\partial L}{\partial w_{k-1,l',l''}} &= \frac{\partial L}{\partial v_{k,l''}} \cdot \frac{\partial v_{k,l''}}{\partial w_{k-1,l',l''}} \\
&= \frac{\partial L}{\partial v_{k,l''}} \cdot \frac{\partial v_{k,l''}}{\partial u_{k,l''}} \cdot \frac{\partial u_{k,l''}}{\partial w_{k-1,l',l''}} \\
&= \frac{\partial L}{\partial v_{k,l''}} \cdot \frac{\partial f(u_{k,l''})}{\partial u_{k,l''}} \cdot \frac{\partial (v_{k-1,l'} \cdot w_{k-1,l',l''} + b_{k-1,l''})}{\partial w_{k-1,l',l''}} \\
&= \underbrace{\frac{\partial L}{\partial v_{k,l''}} \cdot f'(u_{k,l''})}_{\delta_{k,l''}} \cdot v_{k-1,l'}
\end{aligned} \tag{3.3}$$

Nachdem die Gradienten bezüglich der Parameter zwischen der letzten und der vorletzten Schicht berechnet worden sind, werden die Gradienten bezüglich der Parameter zwischen der vorletzten Schicht $\$_{K-1}$ und der Schicht $\$_{K-2}$ betrachtet. In der folgenden Gleichung wird der Gradient der Verlustfunktion L in Bezug auf das Gewicht $w_{k-2,l,l'}$ zwischen den Neuronen l und l' berechnet (siehe Abbildung 3.4). Hierbei entsprechen $u_{k-2,l}$ und $u_{k-1,l'}$ die Werte der Neuronen l und l' vor der Aktivierung und $v_{k-2,l}$ und $v_{k-1,l'}$ die Werte nach der Aktivierung.

$$
\begin{aligned}
\frac{\partial L}{\partial w_{k-2,l,l'}} &= \frac{\partial L}{\partial v_{k,l''}} \cdot \frac{\partial v_{k,l''}}{\partial w_{k-2,l,l'}} \\
&= \frac{\partial L}{\partial v_{k,l''}} \cdot \frac{\partial v_{k,l''}}{\partial u_{k,l''}} \cdot \frac{\partial u_{k,l''}}{\partial w_{k-2,l,l'}} \\
&= \delta_{k,l''} \cdot \frac{\partial u_{k,l''}}{\partial w_{k-2,l,l'}} \\
&= \delta_{k,l''} \cdot \frac{\partial u_{k,l''}}{\partial v_{k-1,l'}} \cdot \frac{\partial v_{k-1,l'}}{\partial w_{k-2,l,l'}} \\
&= \delta_{k,l''} \cdot \frac{\partial u_{k,l''}}{\partial v_{k-1,l'}} \cdot \frac{\partial v_{k-1,l'}}{\partial u_{k-1,l'}} \cdot \frac{\partial u_{k-1,l'}}{\partial w_{k-2,l,l'}} \\
= \delta_{k,l''} \cdot \frac{\partial (v_{k-1,l'} \cdot w_{k-1,l',l''} + b_{k-1,l'})}{\partial v_{k-1,l'}} &\cdot \frac{\partial f(u_{k-1,l'})}{\partial u_{k-1,l'}} \cdot \frac{\partial (v_{k-2,l} \cdot w_{k-2,l,l'} + b_{k-2,l})}{\partial w_{k-2,l,l'}} \\
&= \underbrace{\delta_{k,l''} \cdot w_{k-1,l',l''} \cdot f'(u_{k-1,l'})}_{\delta_{k-1,l'}} \cdot v_{k-2,l'}
\end{aligned}
\tag{3.4}
$$

Bis der Anfang des Netzes erreicht wird, wiederholt sich der Prozess rückwärts. Das Rückwärtslaufen zur Gradientenberechnung ist ein besonderes Merkmal der Rückpropagierung. Durch die Rückpropagierung lässt sich der Einfluss der aktuellen Parameter auf alle Zwischenwerte bis zu der Ausgabe und dem Verlust des Netzes betrachten. Vor allem entspricht die Wiederverwendung der Zwischenergebnisse δ der höheren Schichten dem Ansatz der dynamischen Programmierung und führt zu einer optimalen Berechnung. Nachdem die Gradienten der Verlustfunktion im Bezug auf die Parameter berechnet wurden, lassen sich die optimierten Parameter anhand der Funktion 3.2 berechnen.

3.3 Autoencoder

Die Idee des Autoencoders wurde im Jahr 1987 von Lecun eingeführt [38]. Ein Autoencoder ist ein spezielles, neuronales Netz und besteht aus einem Encoder und einem Decoder. Durch das gleichzeitige Trainieren beider Netze, soll der Encoder die Daten in ihre komprimierte Repräsentation überführen und der Decoder die eingebetteten Daten in ihre Ursprungsformen rekonstruieren. Dabei lässt sich das Konzept des Autoencoders wie folgt darstellen:

$$\begin{aligned} e_\theta(x) &= s_e(Wx + b_e) \\ d_{\theta'}(e_\theta(x)) &= s_d(W'e(x) + b_d) \end{aligned} \tag{3.5}$$

wobei $e_\theta(\cdot)$ dem Encoder mit $e : \mathbb{R}^d \to \mathbb{R}^w$, $d_{\theta'}(\cdot)$ dem Decoder mit $d : \mathbb{R}^w \to \mathbb{R}^d$, $s_e(\cdot)$ die Aktivierungsfunktion des Encoders und $s_d(\cdot)$ die Aktivierungsfunktion des Decoders entsprechen. Dabei stellen $\theta = \{W, b_e\}$ die Parameter des Encoders, $\theta' = \{W', b_d\}$ die Parameter des Decoders und $w << d$ die Dimensionalität des eingebetteten Raumes und der ursprünglichen Trainingsmenge dar.
Schließlich wird der Autoencoder anhand der Minimierung des Rekonstruktionsfehlers trainiert. Formal lässt sich das Minimierungsproblem anhand der Funktion 3.1 wie folgt beschreiben:

$$\theta^*, \theta'^* = \arg\min_{\theta,\theta'} \; L(x, d_{\theta'}(e_\theta(x))) \tag{3.6}$$

wobei $L : \mathbb{R}^r \times \mathbb{R}^r \to \mathbb{R}$ den Rekonstruktionsverlust zwischen den ursprunglichen Daten und den rekonstruierten Daten darstellt. Als die Verlustfunktion kann die mittlere, quadratische Abweichung (engl. mean squared error (MSE)) gewählt werden. MSE misst den Durchschnitt der Quadrate der Rekonstruktionsfehler und ist wie folgt definiert:

$$L(x, d(e(x))) = \frac{1}{n} \sum_{i=1}^{n} \|x - d(e(x))\|_2^2 \tag{3.7}$$

wobei $\|\cdot\|_2$ die euklidische Norm darstellt. Das (Mini-Batch) Gradientenverfahren oder das stochastische Gradientenverfahren kann verwendet werden, um das Optimierungsproblem aus 3.6 zu lösen [61].

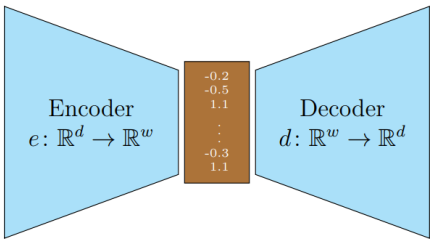


Abbildung 3.5: Illustrierungen des Autoencoders

3.4 Stacked Autoencoder (Deep Autoencoder)

Statt nur eine Zwischenschicht (*Hidden Layer*) wie beim traditionellen Autoencoder zu betrachten, ist in vielen Anwendungen die Arbeit auch mit mehreren Zwischenschichten erwünscht. Vorallem liegt das Interesse auf die schrittweise Verkleinerung der Dimensionen bei den Zwischenschichten des Encoders und die schrittweise Vergrößerung der Dimensionen bei den Zwischenschichten des Decoders. Die Optimierung der Gewichte ist aufwendiger, falls ein nicht-linearer Autoencoder mit mehreren Zwischenschichten betrachtet wird. Insbesondere liegt das Problem an der Bestimmung der Initialgewichte und führt zur Notwendigkeit des separaten Vortrainings [27].

In der Abbildung 3.6 wird die Idee des Vortrainings darstellt. Zuerst wird der erste Autoencoder mit der ersten Zwischenschicht nach der Minimierung der Rekonstruktionsfehler trainiert. Die erhaltenen, eingebetteten Daten der ersten Zwischenschicht werden für das Trainieren des nächsten Autoencoders mit der nächsten Zwischenschicht weiterverwendet. Dieser Vorgang wird wiederholt, bis das Training abgeschlossen ist [42]. Nach dem Vortraining startet die fine-tuning-Phase, wobei der insgesamte Rekonstruktionsfehler mit den Initialgewichten aus dem Vortraining minimiert wird [42].

Zusammengefasst ist ein *Stacked Autoencoder* (*Deep Autoencoder*) ein Autoencoder aus mehreren, gestapelten (engl. *stacked*) Schichten des vortrainierten Autoencoders. Jede Ausgabe eines Encoders wird als Eingabe für den nächsten Encoder betrachtet und erstellt folglich einen tieferen Autoencoder mit mehreren Zwischenschichten.

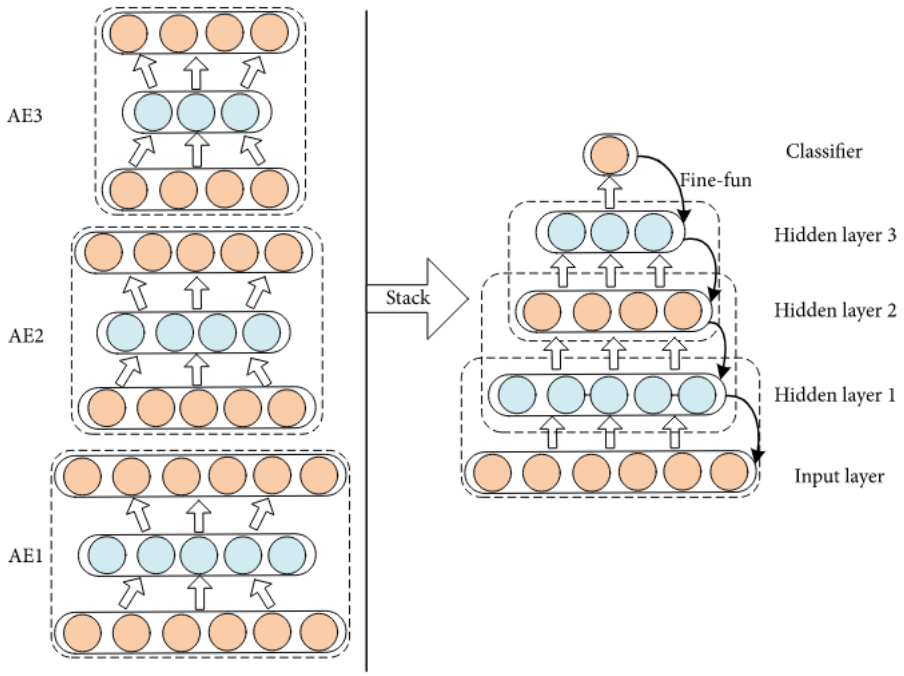


Abbildung 3.6: Struktur des SAEs [42]

3.5 Denoising Autoencoder

Vincent et al. präsentierten im Jahre 2008 den *Denoising Autoencoder* (DAE) [57] als ein erweitertes Konzept des Autoencoders. Das Ziel eines DAEs ist die Rekonstruktion der Eingabe aus ihrem beschädigten Zustand zu ihrem reparierten, ursprünglichen Zustand [57]. Der Fokus verschiebt sich damit von der reinen Rekonstruktion auf die Robustheit des Netzes gegen die Perturbation der Daten. Die Idee der Entrauschung (engl. denoising) der künstlich verfälschten Daten lässt sich anhand der Abbildung 3.7 illustrieren. Hierbei stellen die Punkte x die ursprünglichen Trainingsdaten und die Punkte $\tilde{x}$ die perturbierten Daten aus dem Korruptionsprozess (Dropout) $q_{\mathcal{D}}(\tilde{x}|x)$ dar. Der Korruptionsprozess ist dabei eine stochastische Abbildung für die Umwandelung der Eingabe in ihre perturbierte Form (Kapitel **??**). Das Ziel der Entrauschung ist eine Abbildung von den perturbierten Daten zurück auf die ursprünglichen Versionen zu erstellen. Diese Abbildung lässt sich auch anhand der Manifold-Annahme (egl. *Manifold assumption*) [14] erklären. Die Manifold-Annahme besagt, dass die hoch-dimensionalen Daten ungefähr in der Nähe eines nicht-linearen, niedrig-dimensionalen Manifolds (Mannigfaltigkeit) liegen [14] [58]. Im Bezug auf die Abbildung 3.7 sind die zwei-dimensionalen Daten um den nicht-linearen, eindimensionalen Manifold verteilt. Nun werden die Daten x mit dem Korruptionsprozess perturbiert und befinden sich mit größerer Wahrscheinlichkeit außerhalb des Manifolds. In der Trainingsphase versucht der DAE die perturbierten Daten wieder auf ihre ursprünglichen Positionen in der Nähe des Manifolds zu bringen, indem er eine stochastische Operation $p(X|\tilde{X})$ lernt (je weiter die Daten vom Manifold liegen, desto niedriger ist die Wahrscheinlichkeit) [58]. Somit lernt der DAE gleichzeitig die Verteilung der urspünglichen Daten und die Struktur des Manifolds [57].
Das insgesamte Vorgehen des DAEs ähnelt an dem vom traditionellen Autoencoder:

$$\begin{aligned} \tilde{x} &\sim q_{\mathcal{D}}(\tilde{x}|x) \\ z &= s_e(W\tilde{x} + b_e) \\ \tilde{z} &\sim q_{\mathcal{D}}(\tilde{z}|z) \\ x' &= s_d(W'\tilde{z} + b_d) \end{aligned} \tag{3.8}$$

mit s_e als die Aktivierungsfunktion für den Encoder, s_d als die Aktivierungsfunktion des Decoders und $\theta = \{W, W', b_e, b_d\}$ als die Parameter des Netzes [60]. Das Trainieren des DAEs erfolgt ebenfalls mit der Minimierung des Rekonstruktionsfehlers 3.6 anhand des stochastischen Gradientenverfahrens.

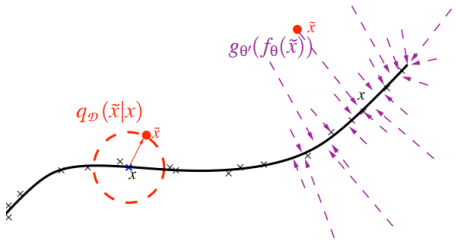


Abbildung 3.7: Visualisierung des Manifold-Lernens mit DAE [57]

Kapitel 4

Deep Embedded Clustering

Eine grundlegende Annahme bei der Datenzusammenfassung ist vor allem, dass ähnliche Ereignisse ähnliche Merkmalsdarstellungen haben. Der Autoencoder bietet zwar eine erfolgreiche Dimensionsreduktion mit der Minimierung der Rekonstruktionsfehler an, jedoch garantiert der Ansatz nicht, dass jede Ähnlichkeit in der hohen Dimension auch in der abgebildeten Dimension vorkommt. Folglich ist ein weiterer Prozess mit dem Fokus auf die Ähnlichkeitserhaltung gewünscht. In diesem Kapitel wird ein Deep-Learning basierter Ansatz für die Clusteringanalyse in der hohen Dimension vorgestellt. Dieser Ansatz versucht die Ähnlichkeit der hohen und der eingebetteten Dimension beizubehalten, indem die Merkmalsrepräsentationen und die optimalen Clustering-Zuordnungen gleichzeitig trainiert werden.

4.1 Clusteringanalyse

Die Clusteringanalyse ist ein verbreiteter, unüberwachter Ansatz im Bereich des maschinellen Lernens für die Datenanalyse und Datenvisualisierung [60]. Das Ziel der Clusteringanalyse ist die Gruppierung der Daten nach deren Ähnlichkeit, sodass ähnliche Daten in dieselbe Gruppe (*Cluster*) zugewiesen werden.

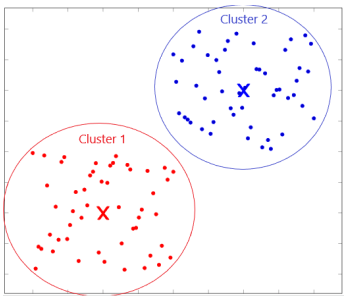


Abbildung 4.1: Illustrierung des Clusterings mit den jeweiligen Clusterzentroiden X

Gegeben ist also eine Menge $\tau = \{x_1, \dots x_N\}$ von Beobachtungen, eine Anzahl von K und eine Distanzmetrik $\|\cdot\|$. Gesucht ist die Einordnung aller Beobachtungen in den Gruppen $C_1, \dots C_K$, sodass folgendes Minimierungsproblem erfüllt wird [49]:

$$C^* = min_C \sum_{k=1}^{K} N_k \sum_{C(i)=k} \|x_i - \mu_k\|^2 \tag{4.1}$$

wobei μ_k die Clusterzentroide (Mittelwert aller Variablen) des Clusters k darstellt und $N_k = \sum_{i=1}^{N} I(C(i) = k)$ gilt. Dabei entspricht $C(i) = k$ einer Abbildung, welche der i-ten Beobachtung das k-te Cluster zuweist. Gewünscht ist hierbei den Abstand zwischen Beobachtungen desselben Clusters zu minimieren und den Abstand zwischen den unterschiedlichen Clustern zu maximieren [49].

Der k-Means-Algorithmus, vorgestellt von MacQueen im Jahre 1967 [45], ist ein bekanntes Verfahren, um das Minimierungsproblem aus 4.1 iterativ zu lösen.

Algorithmus 2 k-Means-Algorithmus [49]

Eingabe: τ, K

Ausgabe: $C_1, \dots, C$

1. Wähle K Punkte aus τ zufällig als Clusterzentroide $\mu_1, \dots, \mu_k$ aus
2. Berechne das Clustering anhand der Zentroide:
   $C(i) = argmin_{1 \leq k \leq K} \|x_i - \mu_k\|_2^2$
3. Aktualisiere die Zentroide für jede C_k:
   $\mu_k^{'} = argmin_{\mu_k} \sum_i^N \|x_j - \mu_k\|_2^2$ mit $x_j \in C_k$
4. Wiederhole Schritt 2 und 3 bis die Zuweisungen sich nicht mehr ändern.

4.2 Deep Embedded Clustering

Auffällig ist jedoch bei der Clusteranalyse und insbesondere dem k-Means-Algorithmus, dass die Ähnlichkeitsberechnung auf die Distanzberechnung beruht. Folglich ist die Analyse (unter der Distanzberechnung) besonders vom Fluch der Dimensionalität betroffen. Des Weiteren stellt sich heraus, dass ein traditionelles Clustering-Verfahren anhand der euklidischen Distanz eine schwache Aussagekraft auf die Rohdaten zeigt [60]. Xie et al. stellen in [60] das *Deep Embedded Clustering* (DEC) Verfahren vor, um das Problem der hohen Dimensionalität in der Clusteringanalyse zu behandeln. DEC ist ein Deep-Learning basierter Ansatz, welcher gleichzeitig die Merkmalsrepräsentationen und die optimalen Clustering-Zuordnungen lernt. Dabei versuchen die Autoren zunächst mit einer nicht-linearen Abbildung $f_\theta : X \to Z$, wobei θ die trainierbaren Parameter repräsentieren, die ursprüngliche Dimensionalität X zu reduzieren. Tiefe, neuronale Netze dienen wegen ihrer theoretischen Eigenschaft der Funktionsannäherung (engl. *theoretical function*

approximation property) [29] [60] und ihrer Fähigkeit zum Erlernen von Merkmalen (engl. *feature learning*) [6] [60] als ein geeignetes Verfahren zur Umsetzung der nicht-linearen, parametrisierten Abbildung f_θ. Schließlich wird das Clusteringproblem in der eingebetteten Dimension Z gelöst, indem die Clusterzentroide und die Parameter θ Schritt für Schritt optimiert und verfeinert werden. Insgesamt lässt sich der gesamte Prozess in zwei Hauptphasen unterteilen:

1. Parameter-Initialisierungsphase: mit Deep Autoencoder,
2. Parameter-Optimierungsphase: durch die Iteration zwischen der Berechnung einer Hilfsverteilung und der Minimierung der Kullback-Leibler-Divergenz.

4.2.1 Parameter-Initialisierungsphase

Initialisiert wird DEC mit einem SAE. Aktuelle Forschungen haben gezeigt, dass der tiefe Autoencoder (SAE) semantisch aussagekräftige und gut getrennte Repräsentationen auf realen Datensätzen erzeugt [58] [27] [37] [60]. Folglich erleichtert er das zusätzliche Lernen von Clusterung-Repräsentation und dient somit als eine geeignete, initiale Repräsentation für die Parameter-Optimierungsphase [60]. Der SAE wird anhand der Minimierung des Rekonstruktionsfehlers aus der Gleichung (3.6) trainiert. Nach dem Training werden nur die Encoder-Schichten betrachtet, um eine Abbildung f_θ des ursprünglichen Datenraums auf den eingebetteten Merkmalsraum zu erschaffen. Schließlich wird der k-Means-Clustering auf die Daten im eingebetteten Raum angewendet, um die k initialen Clusterzentroide für die zweite Phase zu erhalten. Die Abbildung 4.3 illustriert die vorgestellten Schritte der Initialisierungsphase.

4.2.2 Parameter-Optimierungsphase

Mit der initialen Abbildung f_θ und den initialen Clusterzentroiden aus der Initialisierungsphase, werden folglich die Optimierungsschritte durchgeführt. Dabei werden f_θ und die Zentroide anhand des unüberwachten Lernalgorithmus verfeinert. Die folgende zwei Schritte wiederholen sich bis ein Konvergenzkriterium getroffen wurde [60].

1. Berechnung der „weichen Verteilung“ (engl. $soft\ assignment$) zwischen den eingebetteten Daten und den Clusterzentroiden (Die Verteilung stellt die Zuordnungswahrscheinlichkeit der einzelnen, eingebetteten Beobachtungen zu einem Cluster dar).
2. Optimierung der Abbildung f_θ und der Clusterzentroide durch die Anpassung der weichen Verteilung zu der Hilfszielverteilung (Anhand der Minimierung der Kullback-Leibler-Divergenz).

Zunächst werden einige Grundlage zur t-verteilten stochastischen Nachbarschaftseinbettung (t-SNE) (engl. *t-distributed stochastic neighbor embedding*) vorgestellt, wovon der DEC seine Inspiration bekam [44].

Exkurs: Grundlage der t-verteilten stochastischen Nachbarschaftseinbettung

T-SNE ist ein Verfahren, welches die nicht-lineare Dimensionsreduktion ermöglicht. Die Ähnlichkeit zweier Beobachtungen x_i und x_j wird als die Zuordnungswahrscheinlichkeit $p_{i,j}$, dass x_j die Beobachtung x_i als sein Nachbar auswählen würde, betrachtet. Insbesondere folgt die Zuordnungsverteilung der Gauß-Verteilung mit dem Erwartungswert x_i. Die Abbildung 4.2 stellt das Konzept bildlich dar. Zunächst wird eine Gauß-Verteilung mit dem Erwartungswert x_i erzeugt. Die euklidische Distanz zwischen x_i und x_j wird berechnet und auf der X-Achse abgebildet. Folglich stellt die Y-Achse die Zuordnungswahrscheinlichkeit $p_{i,j}$ dar. Den Beobachtungen mit geringer Distanz werden hohe Zuordnungswahrscheinlichkeit zugeteilt und denen mit größeren Distanzen eine infinitesimale Wahrscheinlichkeit [44] [20].

Das Ziel der t-SNE ist, die Beobachtungen in dem eingebetteten, niedrig-dimensionalen Raum so zu platzieren, dass die Nachbarschaftsbeziehung im ursprünglichen, hoch-dimensionalen Raum optimal erhalten bleibt. Demgemäß wird versucht, die Zuordnungsverteilung des ursprünglichen Raums und des eingebetteten Raums zu vergleichen und möglichst gleich anzupassen. Die Zuordnungswahrscheinlichkeiten der beiden Räume lassen sich wie folgt darstellen [44]:

$$p_{i,j} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k \neq l} \exp(-\|x_k - x_l\|^2/2\sigma_i^2)} \tag{4.2}$$

$$q_{i,j} = \frac{(1 + \|y_i - y_j\|^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2/\alpha)^{-\frac{\alpha+1}{2}}} \tag{4.3}$$

wobei $p_{i,j}$ die Zuordnungswahrscheinlichkeit der Beobachtung x_i zu x_j im ursprünglichen Raum und $q_{i,j}$ die Zuordnungswahrscheinlichkeit der Beobachtung y_i zu y_j im eingebetteten Raum darstellt. Die Variable σ_i repräsentiert die Varianz der Gauß-Verteilung mit dem Erwartungswert x_i. Die Varianz σ_i bestimmt vor allem die Größe des Nachbarschaftsraums, weswegen in dichten Regionen eine kleine Varianz angemessener ist [44]. Auffällig ist jedoch, dass die Zuordnungsverteilung Q_i des eingebetteten Raums nicht der Gauß-Verteilung folgt, sondern der studentschen t-Verteilung mit dem Freiheitsgrad von 1 ($\alpha = 1$). Die student-

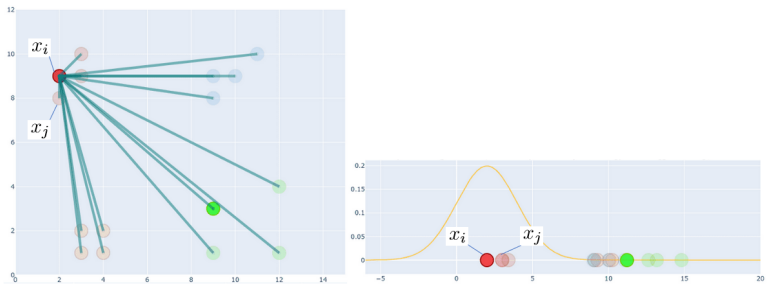


Abbildung 4.2: Illustierung der Zuordnungswahrscheinlichkeit $p_{i,j}$ mit x_i als den Erwartungswert der Zuordnungsverteilung und x_j als einen Nachbar von x_i [20]

sche t-Verteilung „fällt“ schneller aber ihre Ränder konvergieren langsamer gegen 0, sodass die Clustern nicht in einer einzigen Stelle zerdrückt werden (*Crowding Problem*) [44] [20]. Um die Unterschiedlichkeit zwischen zwei Wahrscheinlichkeitsverteilungen zu messen, eignet sich die Verwendung der Kullback-Leibler-Divergenz. Statt die Summe der KL-Divergenzen zwischen allen P_i Verteilungen (zentriert auf x_i) und Q_i Verteilungen (zentriert auf y_i) zu minimieren [28], minimiert T-SNE eine einzelne KL-Divergenz zwischen zwei multivariaten Verteilungen, nämlich der Verteilung P des ursprünglichen Raums und Q des eingebetteten Raums [44].

Schritt 1 der Optimierungsphase: Berechnung der weichen Verteilungen

DEC nutzt ebenfalls die Zuordnungsverteilung, um die Ähnlichkeit der Beobachtungen zu repräsentieren. Der Fokus liegt jedoch nicht mehr auf der Zuordnungswahrscheinlichkeit der einzelnen Beobachtungen zu jeder anderen Beobachtung, sondern auf der Zuordnungswahrscheinlichkeit der einzelnen Beobachtungen zu jedem Clusterzentroid (eine weiche Verteilung). Mit der initialen Abbildung $f_\theta : X \to Z$, den k initialen Zentroiden $\{\mu_j\}_{j=1}^{k}$ und der Gleichung aus (4.3) lässt sich die Zuordnungswahrscheinlichkeit der i-ten Beobachtung zum j-ten Cluster wie folgt darstellen:

$$q_{i,j} = \frac{(1 + \|z_i - \mu_j\|^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'}(1 + \|z_i - \mu_{j'}\|^2/\alpha)^{-\frac{\alpha+1}{2}}} \tag{4.4}$$

mit dem Freiheitsgrad $\alpha = 1$ und $z_i = f_\theta(x_i) \in Z$ als die eingebettete Beobachtung von $x_i \in X$ [60].

Schritt 2a der Optimierungsphase: Hilfsverteilung

T-SNE sucht nach der eingebetteten Repräsentation, welche die Differenz zwischen der Verteilung P des ursprünglichen Raums und Q des eingebetteten Raums minimiert. DEC nutzt die zentroidbasierte, weiche Verteilung Q, jedoch ist die Erstellung der zentroidbasierten Verteilung P problematisch, da keine Clusterzentroide im ursprünglichen Raum bekannt sind. Folglich wird eine Hilfsverteilung benötigt, welche (1) die Clustering-Vorhersage (anhand der Evaluation des Clustering-Ergebnisses) verstärkt, (2) mehr Bedeutung auf die sicheren Beobachtungen mit sehr hoher Zuordnungswahrscheinlichkeit legt und (3) den Verlustbeitrag einzelner Zentroiden normalisiert. Somit wurde beispielsweise die folgende Hilfsverteilung von den Autoren ausgewählt [60]:

$$p_{i,j} = \frac{q_{i,j}^2/f_j}{\sum_{j'} q_{i,j'}^2/f_{j'}} \tag{4.5}$$

wobei $f_j = \sum_i q_{i,j}$ darstellt. Auffallend ist bei der Berechnung von (4.5) die direkte Abhängigkeit zur Verteilung Q. Der Grund ist die Annahme, dass die meisten aus der initialen Abbildung resultierten Beobachtungen, die eine sehr hohe Zuordnungswahrscheinlichkeit

$q_{i,j}$ mit den initialen Clusterzentroiden zugewiesen wurden, eine korrekte Clusterzuordnung liefern [60].

Schritt 2b der Optimierungsphase: Minimierung der KL-Divergenz

Schließlich werden die Parameter θ und die Clusterzentroide anhand des stochastischen Gradientenverfahrens gemeinsam optimiert. Als die Verlustfunktion für die Optimierung wird die KL-Divergenz zwischen der weichen Verteilung Q und der Hilfsverteilung P ausgewählt [60]:

$$L = KL(P\|Q) = \sum_i \sum_j p_{i,j}\, log\, \frac{p_{i,j}}{q_{i,j}} \tag{4.6}$$

Die Gradienten der Funktion 4.6 werden in Bezug auf die einzelnen, eingebetteten Beobachtungen z_i und auf die einzelnen Clusterzentroide μ_j wie folgt berechnet [60]:

$$\frac{\partial L}{\partial z_i} = \frac{\alpha+1}{\alpha} \sum_j (1 + \frac{\|z_i - \mu_j\|^2}{\alpha})^{-1} \times (p_{i,j} - q_{i,j})(z_i - \mu_j), \tag{4.7}$$

$$\frac{\partial L}{\partial \mu_j} = -\frac{\alpha+1}{\alpha} \sum_i (1 + \frac{\|z_i - \mu_j\|^2}{\alpha})^{-1} \times (p_{i,j} - q_{i,j})(z_i - \mu_j) \tag{4.8}$$

Die Gradienten $\partial L/\partial z_i$ werden dann an das DNN weitergegeben und mit der Rückpropagierung werden die Parameter θ optimiert [60].

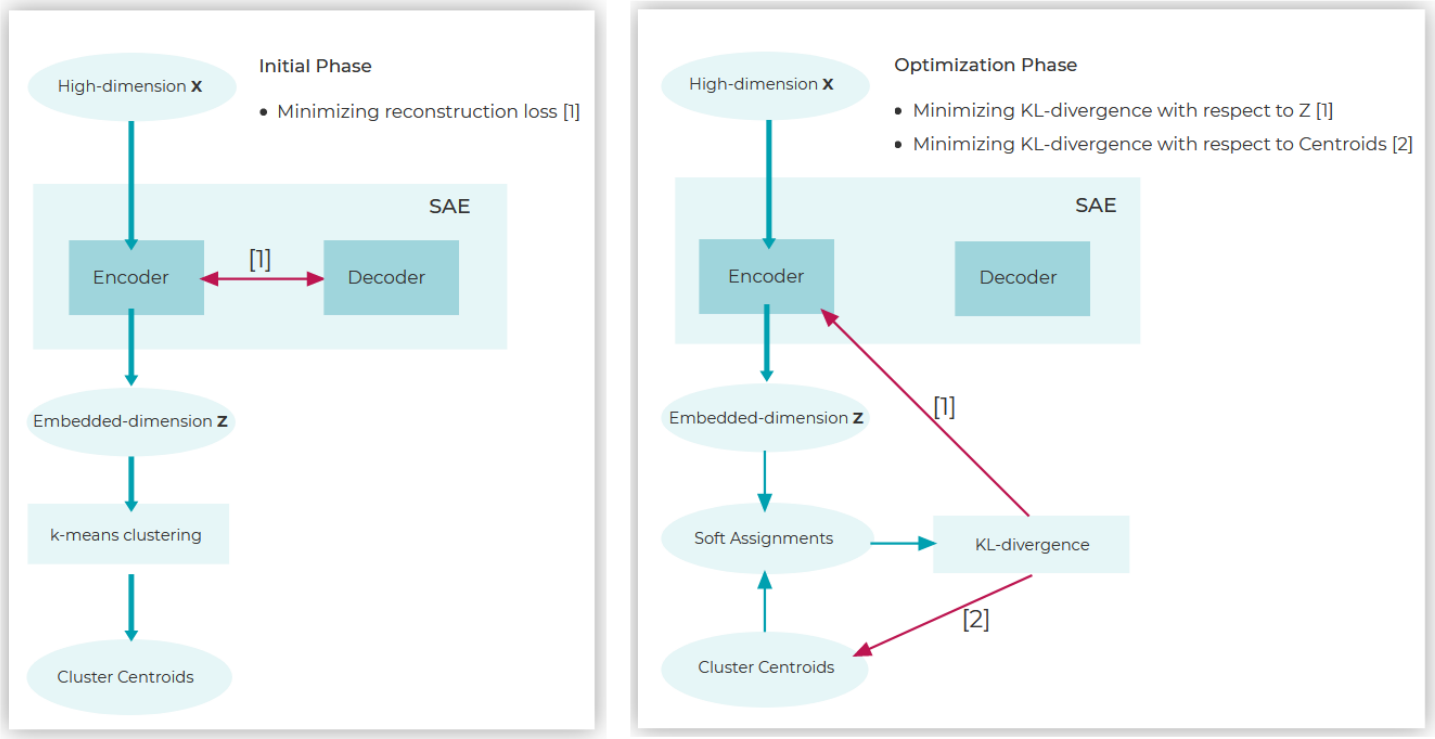


Abbildung 4.3: Illustrierung des Verfahrens DEC

Kapitel 5

Deep Submodular Autoencoder

Die Clusteringanalyse stellt sich als eng verwandt mit dem Datenzusammenfassungsproblem dar. Einen großen Unterschied macht die Auswahl der Zentroide. Die Clusteringanalyse erlaubt künstlich erzeugte Zentroide, wobei in einer Datenzusammenfassung nur existierende, wahre Beobachtungen als Zentroide gewünscht sind. Eine mögliche Lösung ist, statt ein Zentroid selbst als den Repräsentant zu betrachten, eine wahre Beobachtung für jedes Cluster in die Zusammenfassung zu nehmen. Unklar ist jedoch, welche wahre Beobachtungen relevant und geeignet sind und wie das Auswahlkriterium konkret aussieht. Im ungünstigsten Falle wird eine Beobachtung ausgewählt, welche das zugehörige Cluster am wenigsten gut repräsentiert. Die einfachste Auswahlmöglichkeit ist die nächsten Beobachtungen von den jeweiligen Zentroiden zu nehmen (Gleichung 5.4). Jedoch sichert der Ansatz keine hohe Diversität unter den ausgewählten Repräsentanten, sodass eine wesentliche Eigenschaft einer optimalen Zusammenfassung nicht erfüllt wird. Aus diesem Grund erschwert sich die Herangehensweise an das Datenzusammenfassungsproblem. Insgesamt lässt sich die Idee des DECs, bezogen auf das Datentzusammenfassungsproblem, durch die vier vereinfachten Gleichungen aus (3.6), (4.1), (4.6) darstellen:

$$\arg\min_{e,d} L(e,d) \tag{5.1}$$

$$\arg\min_{C} \sum_{k=1}^{K}\sum_{i=1}^{N} I(C(i)=k) \sum_{C(i)=k} \|x_i - \mu_k\|_2^2 \tag{5.2}$$

$$\arg\min_{e,C} KL(P\|Q) \tag{5.3}$$

$$S \leftarrow min_{x_i}(\|x_i - \mu_k\|_2^2)_{i=1}^{N} \text{ FOR ALL } \mu_k \text{ IN } C \tag{5.4}$$

wobei die ersten beiden Gleichungen die Parameter-Initialisierungsphase, die dritte Gleichung die Parameter-Optimierungsphase und die letzte Gleichung eine mögliche Auswahlkriterium der Reräsentanten für die Erstellung einer Zusammenfassung repräsentieren.

Nichtsdestotrotz bietet DEC einige wichtige Erkenntnisse, die für die Erstellung einer Zusammenfassung verwendet werden können. Untersucht wird in diesem Kapitel der Deep Submodulare Autoencoder (DSA) als eine erweiterte Vorgehensweise, um eine optimale Datenzusammenfassung für unstrukturierte, hoch-dimensionale Datensätze zu erstellen.

5.1 Einführung der Submodularität

Obwohl das Clustering-Problem dem Zusammenfassungsproblem ähnelt, führt die Auswahl eines optimalen Repräsentants innerhalb eines Clusters zu einer eigenen Herausforderung. Demzufolge wird die Submodulariät eingeführt. Die submodulare Maximierung unter der Kardinalitätsbeschränkung ($\arg\max_{S\subseteq V,|S|\leq K} f(S)$) mit Hilfe der vorgestellten Approximationsalgorithmen führt zur Erstellung einer guten Zusammenfassung. Insbesondere wird bei der Erstellung gezielt auf die Abdeckung der Repräsentation von den gesamten Daten und auf das Erreichen möglichst hoher Diversität geachtet. Die Anwendung der Submodularität stellt insgesamt im Vergleich zum herkömmlichen Clusteringverfahren eine natürlichere Vorgehensweise zur Auswahl der Repräsentanten. Folglich kann der Datenzusammenfassungsalgorithmus, beispielsweise der gierige Algorithmus, für die initiale Clusterzuordnung am Ende der Parameter-Initialisierungsphase eingesetzt werden:

$$\arg\min_{e,d} L(e,d) \tag{5.5}$$

$$\arg\max_{S} f(S) \tag{5.6}$$

$$\arg\min_{e,S} KL(P\|Q) \tag{5.7}$$

5.2 Gemeinsame Optimierung des Rekonstruktionsfehlers

Eine weitere Unsicherheit des DECs kann in der Parameter-Optimierungsphase auftreten. Nachdem der Autoencoder nach der Minimierung des Rekonstruktionsfehlers trainiert wurde, werden die initialen Zentroide und die Parameter des Netzes anhand der KL-Divergenz ohne die Erhaltung der Rekonstruktion optimiert. Folglich kommen möglicherweise Verfälschungen in der eingebetteten Dimension vor [24], sodass die Aussagekraft für die weiteren Prozesse im eingebetteten Raum verloren geht. Eine einfache Erweiterung ist, den Rekonstruktionsfehler in der Parameter-Optimierungsphase zusätzlich zu betrachten:

$$\arg\min_{e,d} L(e,d) \tag{5.8}$$

$$\arg\max_{S} f(S) \tag{5.9}$$

$$\arg\min_{e,d,S} L(e,d) + \lambda KL(P\|Q) \tag{5.10}$$

wobei $\lambda \in \mathrm{R}$ eine Regulierungsparamter darstellt.

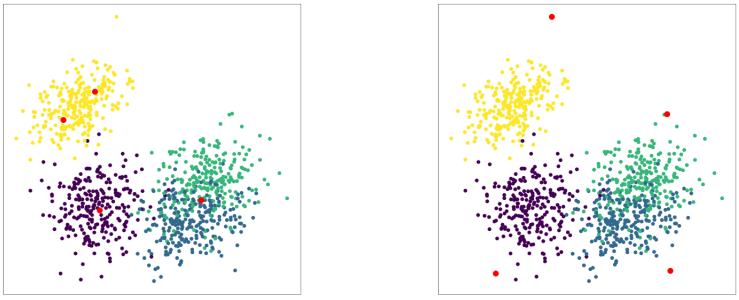


Abbildung 5.1: Vergleich ausgewählter Zentroide zwischen K-Means(links) und Greedy(rechts)

5.3 Weitere Optimierungen

Die größte Schwäche des DECs ist die Annahme, dass der trainierte Autoencoder nach der Parameter-Initialisierungsphase einen aussagekräftigen, eingebetteten Raum erzeugt und möglichst erfolgreiche Initial-Clusterzuordnung ausliefert. Folgich ist die Performanz des Autoencoders und des Clusteringverfahrens entscheidend für den gesamten Erfolg des DECs. Die eingeführte Submodularität aus dem Abschnitt (5.1) kann den gesamten Prozess stabilisieren. Ein Merkmalsraum, in dem jede Ähnlichkeit des ursprünglichen Raums erhalten bleibt, kann durch einen trainierten Autoencoder nicht garantiert werden. Folglich können Beobachtungen fälschlich abgebildet werden (sogenannte „Ausreißern“), sodass ihre Ähnlichkeitsbeziehungen im eingebetteten Merkmalsraum nicht der vom urspünglichen Raum entsprechen. Für das Clusteringverfahren ist dieser Merkmalsraum jedoch entscheidend für eine optimale Clusterzuordnung, da die Berechung der Zentroide empfindlich auf Ausreißern reagieren kann. Die Abbildung 5.1 illustriert ein mögliches Szenario für das Vorkommen von Ausreißern. Zunächst wurden vier eindeutige Cluster mit einer Dimensionalität von 100 erstellt[1]. Anschließend wird der Autoencoder trainiert, um eine Abbildung auf den eingebetteten Raum der Dimensionalität 2 zu realisieren[2]. In der Abbildung ist der eingebettete Raum nach dem Training zu sehen. Auffällig ist vor allem, dass die Cluster der Farbe blau und grün visuell keine klare Trennung zeigen. Nach der genaueren Analyse liegen jedoch die Mittelwerte des jeweiligen Clusters klar voneinander getrennt, sodass eine lineare Entscheidungsgrenze (engl. *Decision boundary*) eingeführt werden kann. Nichtsdestotrotz scheitert das Clusteringverfahren die erwarteten Zentroide auszuwählen, da alle Beobachtungen, einschließlich der Ausreißer, gleich stark behandelt werden. Jedoch besteht das Interesse des Datenzusammenfassungsproblems nicht darin, eine optimale Clusterzuordnung zu erreichen, sondern möglichst unähnliche Beobachtungen, die gleichzeitig eine hohe Aussagekräfte haben, zu finden. Für dieses Interesse liegt der Fokus nicht mehr auf der Erhaltung möglichst aller Ähnlichkeitsbeziehungen, sondern nur auf der Erhaltung der auffälligsten Unähnlichkeiten. Somit erlaubt die Arbeit mit der submodularen Maximie-

[1] SKLEARN.DATASETS.MAKE_BLOBS [16]
[2] trainiert bis zu einem Rekonstruktionsfehler von 0.05

rung einige unvermeidbare Ausreißer. Für die Abbildung 5.1 sind vier Beobachtungen, die am unähnlichsten sind und gleichzeitig den gesamten Datensatz am besten repräsentieren, nämlich die Randpunkte der jeweiligen Cluster.

Beim DEC führen die „schlecht ausgewählten“, initialen Zentroide zu weiteren Misserfolgen in der Verfeinerungsphase. Eine weitere, mögliche Erweiterung ist, die starke Abhängigkeit der Parameter-Optimierungsphase zu den initialen Zentroiden abzuschaffen, indem die Zentroide bei jeder Iteration erneut berechnet werden. Mit den vorhandenen, schnellen Datenzusammenfassungsalgorithmen, wie der ThreeSieves-Algorithmus, kann eine abwechselnde Optimierung zwischen der Minimierung und der Maximierung erfolgen:

$$\arg\min_{e,d} L(e,d) + \lambda KL(P\|Q) \tag{5.11}$$

$$\arg\max_{S} f(S) \tag{5.12}$$

Folglich ist die gesamte Parameter-Initialisierungsphase und die gemeinsame Optimierung nicht mehr erforderlich. Insbesondere verbessert sich die Gesamtleistung durch die gleichzeitige Optimierung der Rekonstruktionsfehler, der Clusterzuordnung und der Zusammenfassung. Nach der Initialisierung einer geschätzten Zusammenfassung S, wird der Autoencoder anhand der Gleichung 5.11 für eine Epoche trainiert. Anschließend werden die Repräsentanten der Zusammenfassung anhand der Gleichung 5.12 erneut ausgewählt und das Training des Autoencoders für die nächste Epoche (Gleichung 5.11) fortgesetzt.

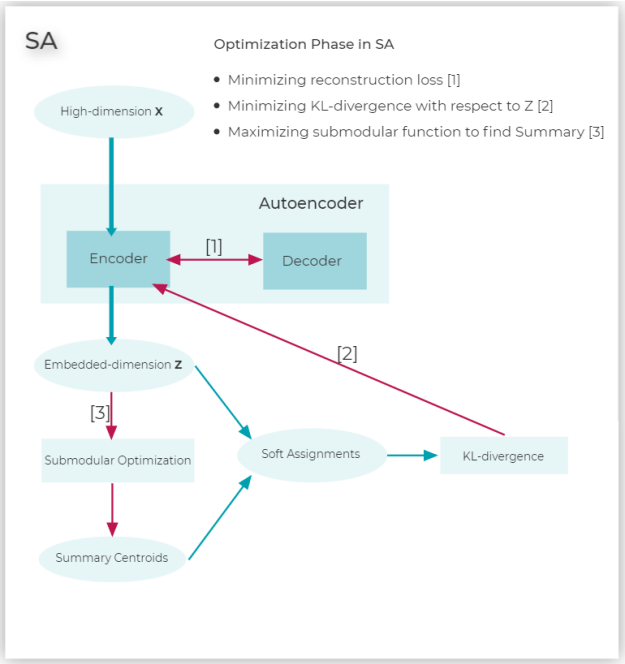


Abbildung 5.2: Illustrierung des Verfahrens DSA

Kapitel 6

Implementierungen und Experimente

Die vorgestellten Verfahren, DEC und DSA, aus den Kapiteln 4 und 5 sollen anschließend durch die kommenden Implementierungen und Experimenten genauer analysiert werden. Zunächst werden im Abschnitt 6.1 zwei Bilddatensätzen vorgestellt, anhand deren die Experimente erfolgen werden. Anschließend werden zwei Metriken zur Evaluation der Ergebnisse eingeführt. Im Abschnitt 6.3 wird das Verfahren DEC bezüglich des Clustering- und Zusammenfassungsproblems untersucht und über mögliche Probleme aus dem Kapitel 5 diskutiert. Schließlich wird im Abschnitt 6.4 eine Implementierungsidee des DSAs vorgestellt und über mögliche Verbesserungen der Implementierung auseinandergesetzt. Alle Experimente sind in der Sprache *Python*[1] programmiert und die Implementierungen der Neuronalen Netzen sind mit dem Module *Pytorch Lightning*[2] realisiert worden. Alle relevanten Codes zu den Experimenten sind im Anhang zu finden.

6.1 Datensätze

Für die vorstehenden Experimente werden die zwei folgenden Bilddatensätze betrachtet.

MNIST: Der MNIST-Datensatz besteht aus 70000 Graustufenbildern von handgeschriebenen Ziffern, wobei jede Ziffer in einer Größe von 28×28 Pixeln gespeichert ist [39]. Der Datensatz enthält 60000 Trainingsbilder und 10000 Testbilder. Es gibt insgesamt 10 Klassen von Ziffern: 0 bis 9 und für alle Bilder sind ihre zugehörige Klassen bekannt (*Labeled Datensatz*).

CIFAR10: Der CIFAR-10-Datensatz besteht aus 60000 Farbbildern der Größe 32×32 Pixel [33]. Insgesamt enthält er 10 verschiedene Klassen: Flugzeug, Auto, Vogel, Katze, Hirsch, Hund, Frosch, Pferd, Schiff und LKW, wobei für jedes Bild seine zugehörige Klasse bekannt ist (*Labeled Datensatz*). Der Datensatz beinhaltet 50000 Trainingsbilder und 10000 Testbilder.

[1] https://www.python.org
[2] https://www.pytorchlightning.ai

Datensatz	Anzahl der Daten	Anzahl der Klassen	Dimensionalität
MNIST [39]	70000	10	28 ×28
CIFAR10 [33]	60000	10	$3 \times 32 \times 32$

Tabelle 6.1: Datensätze MNIST, CIFAR10

DEC und DSA sind unüberwachte Lernmethoden, weshalb Bewertungen auf die Qualität der Ergebnisse schwer umzusetzen sind. Bei MNIST und CIFAR10 handelt es sich jedoch um Labeled Datensätze[3]. Um die Evaluierung zu erleichtern, werden die wahren Klassen (engl. *True label*) nur für die vorzustellenden Evaluierungsmetriken betrachtet und für den gesamten Prozess des Verfahrens als unbekannt angenommen (wie *Unlabeled Datensätze*). Insbesondere wird die klassische Unterscheidung zwischen den Trainings- und Testdaten nicht stattfinden. Das bedeutet, dass die gesamten Daten sowohl die Trainings- als auch die Testdaten darstellen. DEC und DSA dienen im Bezug auf das Datenzusammenfassungsproblem nicht als ein „Vorhersage-Klassifikator“. Das heißt, dass das Interesse nicht auf die Zustandsvorhersage der neuen Daten liegt, sondern auf die Erstellung einer Zusammenfassung der derzeitig vorhandenen Daten. Folglich kann *Overfitting*[4] als mögliches Problem auftreten. Das Problem hat jedoch keinen Einfluss auf das Verfahren, da eine Vorhersage im Kontext der Datenzusammenfassung nicht relevant ist. Ein möglicher umgewandelter Ansatz des DSAs in Bezug auf die Vorhersage könnte seinen Fokus auch auf die Erstellung neuer Zusammenfassung legen, falls ein neues, geeigneteres Element betrachtet wird. Die Vorhersage bezieht sich dann auf die korrekte Abbildung des neuen Elements (bezüglich der Ähnlichkeitserhaltung) auf den eingebetteten Raum. Schließlich muss ein Austausch stattfinden, falls das Hinzügen des neuen Elements in die Zusammenfassung einen besseren Score erzeugt.

[3] Das bedeutet, dass die Klassen für alle Daten bereits vor dem Training bekannt sind.

[4] Das Problem stellt die Überanpassung des Models an den trainierten Daten dar, sodass eine Vorhersage der neuen Daten nicht erfolgen kann.

6.2 Evaluation Metrik

In diesem Abschnitt werden zwei Metriken vorgestellt, die zur Evaluierung der vorgestellten Verfahren dienen. Insbesondere liegt das Interesse an den folgenden zwei Aspekten. Zunächst kann eine gute Zusammenfassung nur erhalten werden, falls möglichst eindeutige Cluster im eingebetteten Raum entstanden sind. Daher ist die erfolgreiche Clusterabbildung für das Datenzusammenfassungproblem ebenfalls von großer Bedeutung. Deshalb wird die *Unsupervised Clustering Accuracy* für den Vergleich und die Evaluierung der Clusteringergebnisse eingeführt. Zusätzlich wird das TSNE-Verfahren auf dem eingebetteten Raum angewendet, um eine visuelle Evaluierung der Clusterzustände zu ermöglichen. Nichtsdestotrotz ist das endgültige Ziel eine diverse, vollständige Zusammenfassung der hoch-dimensionalen Daten zu erhalten. Deswegen wird die *Summary Diversity Metric* eingesetzt, um die Qualität der erhaltenen Zusammenfassung zu prüfen.

Unsupervised Clustering Accuracy (ACC)

Um einen direkten Vergleich zu den Ergebnissen aus Xie et al. [DEC] zu ermöglichen, wird in dieser Arbeit ebenfalls die Unsupervised Clustering Accuracy (ACC) für die Evaluation des Clustering-Ergebnisses eingesetzt. ACC ist eine Evaluationsmetrik für die unüberwachte Clusteringanalyse und lässt sich wie folgt darstellen [60]:

$$ACC = \max_{m} \frac{\sum_{i=1}^{n} \mathbf{1}\{l_i = m(c_i)\}}{n}, \tag{6.1}$$

wobei n die Anzahl der gesamten Daten, l_i die wahre Klasse der i-ten Beobachtung und c_i den zugeteilten Cluster des Clusteringverfahrens an der i-ten Beobachtung darstellen. Die Anzahl der Cluster entsprechen hierbei jeweils der Anzahl der wahren Klassen. Außerdem wird eine Mapping-Funktion m benötigt, um die beste Zuordnung zwischen den zugeteilten Clustern und den wahren Klassen zu finden, da ein unüberwachter Algorithmus eine andere Klasse als die tatsächliche wahre Klasse zur Darstellung desselben Clusters verwenden kann [15]. Die beste Mapping-Funktion kann anhand des *Hungarian Algorithmus* effizient berechnet werden [34] [60]. Eine Python-Implementierung der Metrik ist unter dem folgenden Link[5] verfügbar.

[5] https://github.com/XifengGuo/DEC-keras/blob/master/metrics.py [60]

Summary Diversity Metric (SDM)

Die Bewertung einer Zusammenfassung muss folgende zwei Aspekte berücksichtigen: Die Vollständigkeit und die Diversität. Daher wird die folgende, einfache Gleichung für die Evaluierung des Zusammenfassungs-Ergebnisses eingeführt:

$$SDM = \frac{\sum_{i=1}^{k} \mathbf{1}\{\exists j = \{1, \ldots, k\} : true_i = l_{s_j}\}}{k}, \tag{6.2}$$

wobei k die Größe der Zusammenfassung, l_{s_j} die wahre Klasse des j-ten Elements der Zusammenfassung und $true_i$[6] die i-te Klasse des Labeled Datensatzes darstellen. Vor allem entspricht die Größe der Zusammenfassung der Anzahl der wahren Klassen. Hiermit wird in der Metrik überprüft, ob alle k Klassen eines Datensatzes in der Zusammenfassung vorkommen, sodass die Vollständigkeit und die Diversität gleichzeitig gewährleistet werden.

TSNE als visuelle Darstellung

Der Kapitel 4.2.2 stellt TSNE als ein Verfahren zur Ermöglichung der nicht-linearen Dimensionsreduktion vor. Das Verfahren wird ebenso zum Zweck der Datenvisualisierung eingesetzt. Da insbesondere auf die Erhaltung der Nachbarschaftsbeziehung im ursprünglichen Raum geachtet wird, dient das Verfahren ebenfalls zur Visualisierung der Clusterzustände. In dieser Arbeit werden die Abbildungen, die vom TSNE-Verfahren erzeugt wurden, zur Visualisiernug der Clusterzustände im eingebetteten Raum verwendet. Hierbei liefert das Verfahren eine zwei-dimensionale Abbildung aus dem zehn-dimensionalen, eingebetteten Raum. Die Farben deuten auf die verschiedenen Klassen des Datensatzes, wobei die schwarze Punkte die aktuellen, gespeicherten Zentroide der jeweiligen Epochen darstellen. Vor allem sind die Abbildungen der TSNE von hoher Bedeutung für die Evaluierung der Ergebnisse. In DEC basieren die Berechnungen der oben genannten Metriken auf das Clusteringverfahren. Um die Zwischenzustände in der Initialisierungsphase zu untersuchen, müssen die Zentroide am Ende jeder Epoche neu berechnet werden (mehr dazu siehe Abschnitt [Zusätzliche Anmerkungen]). Folglich ist die Aussagekraft der Zentroid-basierten Metriken nicht besonders zuverlässig. Des Weiteren ist die Evaluierungsmetrik ACC ungeeignet für den DSA, da die Zentroide, die DSA aussucht, keine Clusterzentroide darstellen. Aus diesem Grund erfolgt die Interpretation der Ergebnisse immer in der Kombination mit den TSNE-Abbildungen.

Zusätzliche Anmerkungen

Wie vorhin angedeutet, müssen die Parameter-Initialisierungsphase und Parameter-Optimierungsphase des DECs für die Evaluierung anhand der vorgestellten Metriken unterschiedlich betrachtet werden. Der hauptsächliche Unterschied der beiden Phasen ist die Verfügbarkeit der

[6] *true* ist eine Menge, die alle vorkommende, wahre Klasse eines Labeled Datensatzes enthält.

Zentroide. Die Initialisierungsphase benötigt kein Kenntnis über die aktuellen Zentroiden, da nur nach dem Rekonstruktionsfehler trainiert wird. Jedoch ist die Information der Zentroide für die Evaluation essentiell. da die Zentroide für die Berechnung der ACC- und SDM-Accuracy benötigt werden. Deswegen werden am Ende jeder Epoche während der Initialisierungsphase die Zentroide nur für die Evaluation erneut berechnet. Insbesondere läuft die Berechnung nach dem selben Verfahren ab wie sie vom DEC bei der Berechnung von initialen Zentroiden in der Optimierungsphase abläuft. Das Verfahren zum Berechnen der Zentroide wird im kommenden Abschnitt näher erläutert. Hinter dieser Idee liegt die Fragestellung, wie die initialen Zentroide aussehen würden, falls die aktuelle Epoche die letzte Epoche der Initialisierungsphase entspreche. Für das Zusammenfassungsproblem auf DEC stellt sich die Frage des Auswahlkriteriums (Kapitel 5). In dieser Arbeit bilden die nächsten Beobachtungen der jeweiligen Clusterzentroide die Zusammenfassung.

6.3 Experiment I: Performanz von DEC

In diesem Abschnitt wird das DEC-Verfahren genauer analysiert. Zunächst liegt der Fokus auf die Reproduzierung des Ergebnisses der ursprünglichen DEC-Publikation auf dem MNIST Datensatz [60]. Betrachtet wird das Verfahren ebenfalls auf dem Datensatz CIFAR10. Des Weiteren werden das Verhalten des DECs und die möglichen, vorgestellten Probleme aus dem Kapitel 5 präsentiert.

6.3.1 Nach-Implementierung des originalen DECs

Den Anweisungen der Autoren nach, werden die Netzwerk-Dimensionen auf d-500-500-2000-10 für alle Datensätze gesetzt, wobei d die Dimensionalität der jeweiligen Datensätze darstellen. DEC wird anhand des Stacked Autoencoders mit Dropout/Denoising-Schichten realisiert, wobei alle Schichten des Stacked Autoencoders dicht und vollständig verbunden sind (engl. *Densely fully connected layers*) [60]. Zunächst werden die Daten x anhand dessen Erwartungswertes μ und deren Standardabweichung σ durch die Formel: $(x-\mu)/\sigma$ normalisiert (engl. *Standardization*). Vor dem Training werden die Gewichte des Autoencoders auf einem zufälligen Wert anhand der Gauß-Verteilung (mit einem Erwartungswert von 0 und einer Standardabweichung von 0.01) gesetzt. Für die Parameter-Initialisierungsphase werden die Schichten des Stacked Autoencoders zunächst mit einer Dropout-Rate von 0.2 für 200 Epochen trainert. Beim Dropout werden einige Eingaben mit einer Wahrscheinlichkeit von 20% zufällig auf Null gesetzt[7]. Nach 200 Epochen wird der Stacked Autoencoder für weitere 400 Epochen ohne Dropout-Schichten verfeinert. Als Verlustfunktion wird die MSE Gleichung aus dem Kapitel 3.7 in Bezug auf dem Rekonstruktionsfehler eingesetzt und für die Optimierung des Netzes wird das stochastische Mini-Batch Gradientenverfahren der Größe 256 verwendet. Die Lernrate wird am Anfang auf 0.1 gesetzt und alle 300 Epochen durch 10 geteilt. Hierdurch wird versucht, einen Autoencoder mit einem guten Rekonstruktionsfehler für alle Datensätze zu erreichen. Schließlich werden anhand des k-Means-Algorithmus aus der Gleichung (2) die initialen Zentroide für die nächste Phase berechnet. Insgesamt wird der k-Means-Algorithmus 20-mal unabhängig von einander ausgeführt und die beste Lösung als initiale Zentroide ausgewählt[8]. Das Auswahlkriterium erfolgt durch die Betrachtung der kleinsten *Inertia*. Sie misst den Erfolg des k-Means-Algorithmus für einen Datensatz und wird als die Summe der quadratischen Abstände zwischen jeder Beobachtung und ihrem Zentroid (*Sum of Squared Errors*) dargestellt. Schließlich werden nur die Encoder-Schichten für die nächste Phase weiter betrachtet und trainiert. In der Parameter-Optimierungsphase wird für die Minimierung der KL-Divergenz ebenfalls das stochastische Mini-Batch Gradientenverfahren der Größe 256 als das Optimierungsverfahren gewählt. Das Netz wird mit einer konstanten Lernrate von 0.01 trainiert. Vorgesehen

[7] `https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html`
[8] `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html`

war zusätzlich die Einführung eines Konvergenzkriteriums von 0.1%. In dieser Arbeit wird jedoch das Netz in 100 Epochen vollständig trainiert, um einen besseren Vergleich zwischen den verschiedenen Verfahren zu realisieren.

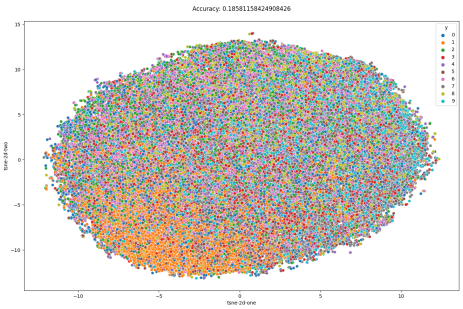
6.3.2 Resultat auf dem Datensatz MNIST

DEC schafft es insgesamt das Clusteringproblem auf MNIST mit einem ACC-Accuracy von 85,53% zu lösen. Dieses Clustering-Ergebnis ist ungefähr 1,2% mehr als das vorgestellte Ergebnis in der ursprünglichen Publikation des DECs [60]. Beim Zusammenfassungsproblem zeigt DEC einen klaren Erfolg mit einem Accuracy von 100%. Die TSNE-Abbildung in 6.1(h) stellt den eingebetteten Raum nach der letzten Epoche des gesamten Prozesses dar. Hierbei repräsentieren die y-Labels die Ziffern 0 bis 9, wobei das Label 10 dem Zentroid entspricht. In der TSNE-Abbildung ist auch zu erkennen, dass es DEC nicht gut gelungen ist, die Ziffern 3 und 5 eindeutig zu unterscheiden.

Diskussion über das Resultat

Das oberste und das unterste Diagramm der Abbildung 6.6 stellen die Accuracy ACC und SDM während des DEC-Verlaufs dar. Bei beiden Diagrammen ist die starke Schwankung des Accuracys auffällig. Bei der Clusteringanalyse zeigt ACC eine Schwankung von ungefähr 5% bis 10% zwischen zwei Epochen[9] und beim Datenzusammenfassungsproblem eine Schwankung von ungefähr 10% bis 30%. Diese Schwankungen deuten auf das bereits erwähnte Problem der Instabilität des Clusteringverfahrens. Das folgende Resultat kann eine deutliche Auswirkung auf den gesamten Prozess haben. Beispielsweise zeigt die Abbildung 6.2 die Folge eines misslungenen Clusteringverfahrens. Hierbei wurden am Anfang der Parameter-Optimierungsphase zwei ungeschickte Zentroide ausgewählt, nämlich ein Zentroid zwischen 4 und 9 und ein anderes Zentroid zwischen 2,3,4,5,6,8 und 9. Folglich entstehen nach der Optimierungsphase zwei unklare Cluster. Es wird außerdem festgestellt, dass die Schwankungen bei beiden Diagrammen aus der Abbildung 6.6 ungefähr ab der 370-ten Epoche stark reduziert werden und kaum zu erkennen sind. Die Abbildung 6.1 könnte den Grund dafür bereitstellen. Sie visualisiert zusätzlich die Veränderung des eingebetteten Raums durch die Parameter-Initialisierungs- und Parameter-Optimierungsphase. Erkennbar ist, dass die Initialisierungsphase ab der Epoche 300 einen guten Clusterzustand im eingebetteten Raum erzeugt. Folglich könnte eine gewisse Stabilität für das Clusteringverfahren eingeführt worden sein. Insbesondere führt diese Stabilität mit sehr hoher Wahrscheinlichkeit zu einem konstanten Erfolg der Erstellung einer optimalen Zusammenfassung.

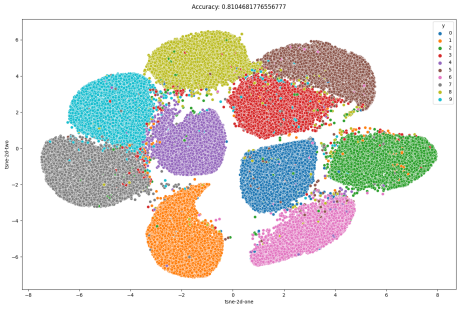
[9] Semantisch deutet diese Schwankung der SDM auf die unterschiedliche Auswahl von 1 bis 3 essenziellen Zentroiden zwischen den Epochen



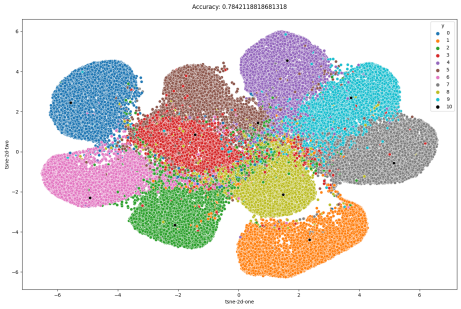
(a) Nach Epoche 2 der Initialisierungsphase mit ACC: 18.58%, SDM: 60%

(b) Nach Epoche 50 der Initialisierungsphase mit ACC: 39.44%, SDM: 70%

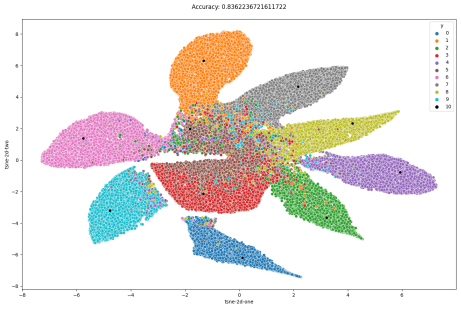
(c) Nach Epoche 300 der Initialisierungsphase mit ACC: 65.93%, SDM: 80%



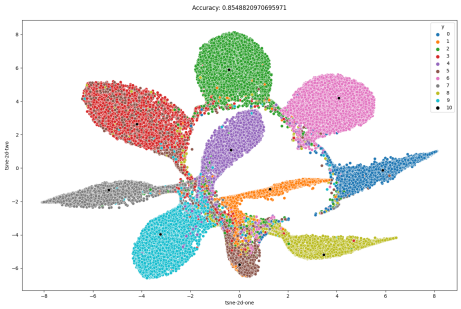
(d) Nach Epoche 599 der Initialisierungsphase mit ACC: 81.05%, SDM: 100%



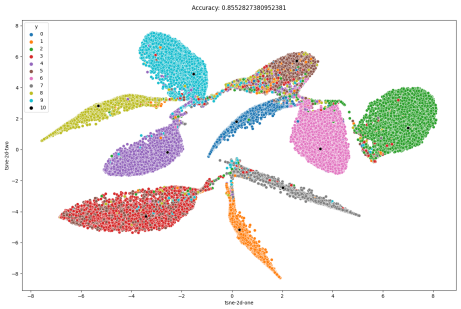
(e) Vor Epoche 0 der Optimierungsphase mit ACC: 78.42%, SDM = 100%



(f) Nach Epoche 2 der Optimierungssphase mit ACC: 83.62%, SDM = 100%



(g) Nach Epoche 50 der Optimierungssphase mit ACC: 85.49%, SDM = 100%



(h) Nach Epoche 99 der Optimierungssphase mit ACC: 85.53%, SDM = 100%

Abbildung 6.1: Illustrierung der Clusteringzustände des eingebetteten Raums nach der Epoche 2, 50, 300, 599 während der Parameter-Initialisierungsphase und vor der Epoche 0 (Initialzustand), sowie nach der Epoche 50, 300, 599 während der Parameter-Optimierungsphase anhand des TSNE Verfahrens [**Datensatz MNIST**]

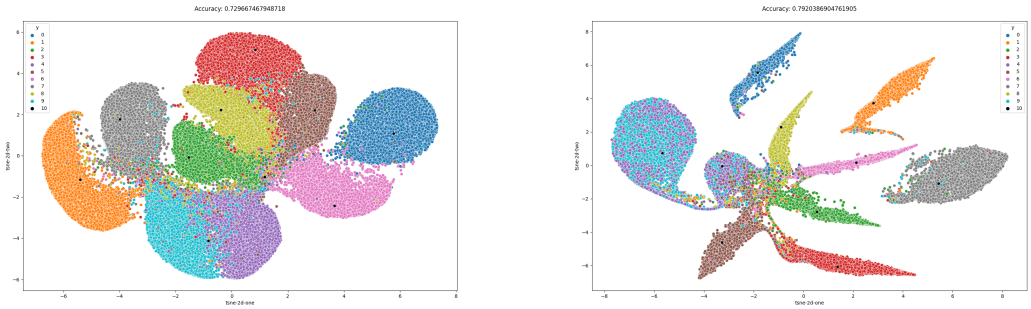


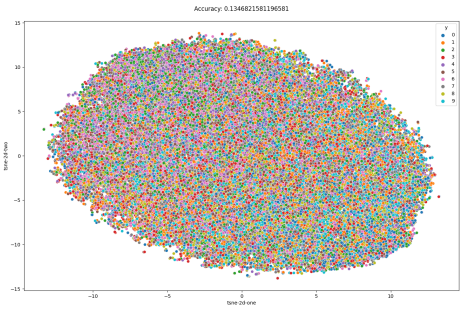
Abbildung 6.2: Illustrierung des eingebetteten Raums anhand des TSNE-Verfahrens für ein misslungenes Clusteringverfahren und seine Auswirkung auf das Endergebnis des DECs. (Linke Abbildung zeigt den eingebetteten Raum und die initialisierten Zentroide kurz vor der Optimierungsphase) (Rechte Abbildung zeigt den eingebetteten Raum nach der Optimierungsphase) [**Datensatz MNIST**]

6.3.3 Resultat auf dem Datensatz CIFAR10

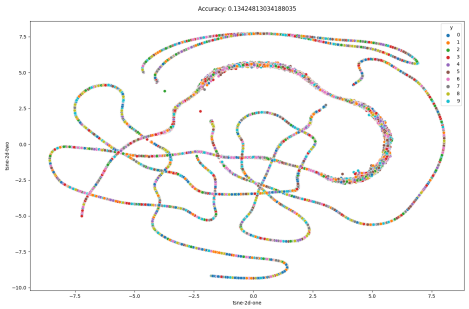
DEC schafft es insgesamt, das Clusteringproblem auf dem Farbbild-Datensatz CIFAR10 mit einem ACC-Accuracy von ungefähr 20% zu lösen. Mit dem vorgestellten Auswahlverfahren für die Zusammenfassung ergibt sich ein SDM-Accuracy von ungefähr 60%. Die TSNE-Abbildung in 6.3(h) stellt den eingebetteten Raum nach der letzten Epoche des gesamten Prozesses dar, wobei die y-Labels folgende Klassen repräsentieren: (0: Flugzeug, 1: Auto, 2: Vogel, 3: Katze, 4: Hirsch, 5: Hund, 6: Frosch, 7: Pferd, 8: Schiff, 9: LKW und 10: Zentroide der jeweiligen Epochen).

Diskussion über das Resultat

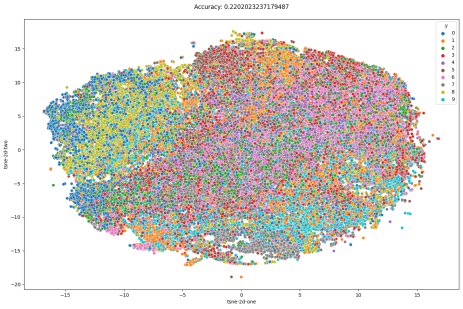
In der Abbildung 6.3(h) sind zwar Clustern zu erkennen, jedoch repräsentieren sie nicht eindeutig jeweils eine Klasse, wobei bei zwei Clustern die Mehrheit leicht zu sehen sind (bei den Klassen Auto und Schiff). Deswegen besitzt der SDM-Accuracy von 60% alleine wenig Aussagekraft, da die ausgewählten Beobachtung der Zusammenfassung visuell keinen eindeutigen Repräsentant des Clusters darstellen.
Starke Schwankungen des Accuracys ACC und SDM sind in den Diagrammen der Abbildung 6.7 ebenfalls zu sehen, wobei das obere Diagramm den ACC-Accuracy und das untere Diagramm den SDM-Accuracy darstellt. Ähnlich wie beim Datensatz MNIST lassen sich die Schwankungen anhand der Instabilität des Clusteringverfahrens begründen. Vor allem ist die Reduzierung der Schwankungen wie beim Datensatz MNIST in den Diagrammen nicht zu erkennen. Anhand der TSNE-Abbildungen 6.3 lässt sich feststellen, dass keine klaren Cluster während des gesamten Prozesses gebildet wurden. Deshalb kann kein stabilisierendes Verhalten des Clusteringverfahrens folgen, und das spiegelt sich in den Accuracies ACC und SDM wieder.



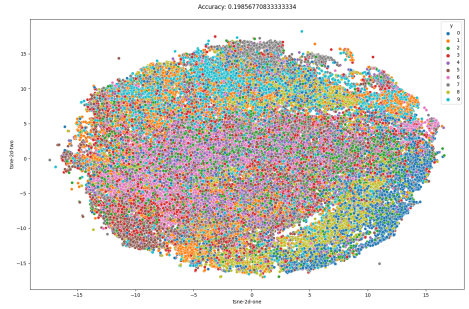
(a) Nach Epoche 0 der Initialisierungsphase mit ACC: 13.47%, SDM: 60%



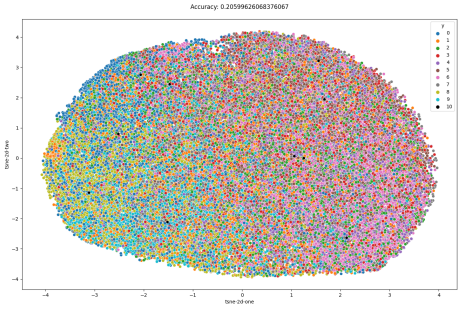
(b) Nach Epoche 10 der Initialisierungsphase mit ACC: 13.42%, SDM: 80%



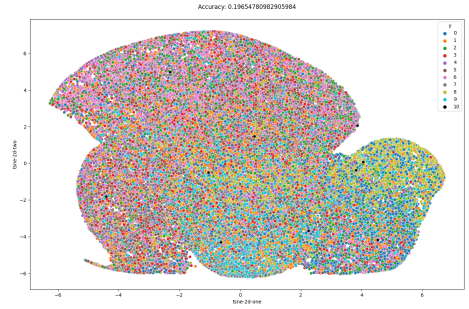
(c) Nach Epoche 300 der Initialisierungsphase mit ACC: 22.02%, SDM: 60%



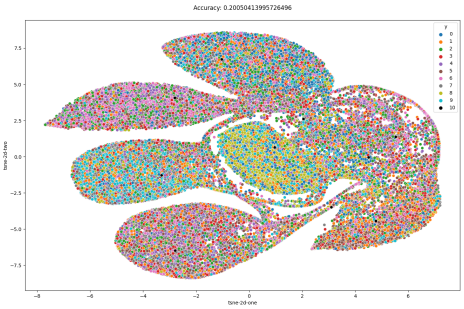
(d) Nach Epoche 599 der Initialisierungsphase mit ACC: 19.86%, SDM: 70%



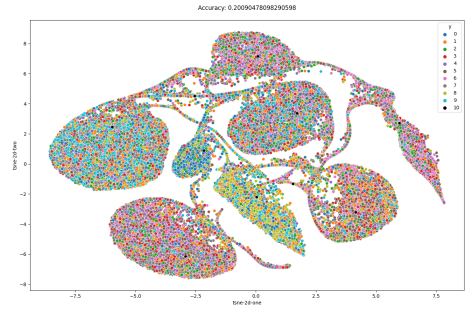
(e) Vor Epoche 0 der Optimierungsphase mit ACC: 20.06%, SDM = 80%



(f) Nach Epoche 5 der Optimierungssphase mit ACC: 19.65%, SDM = 90%



(g) Nach Epoche 50 der Optimierungssphase mit ACC: 20.05%, SDM = 70%



(h) Nach Epoche 99 der Optimierungssphase mit ACC: 20.09%, SDM = 60%

Abbildung 6.3: Illustrierung der Clusteringzustände des eingebetteten Raums anhand des TS-NE Verfahrens nach der Epoche 0, 10, 300, 599 während der Parameter-Initialisierungsphase und vor der Epoche 0 (Initialzustand), sowie nach der Epoche 5, 50, 99 während der Parameter-Optimierungsphase auf dem **Datensatz CIFAR10**

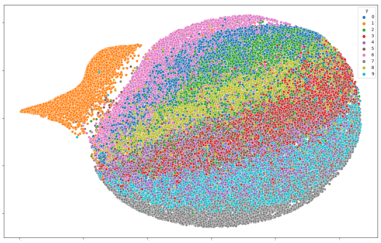
6.3.4 Diskussion über einen besseren Autoencoder

Auf dem Datensatz MNIST hat DEC erfolgreiche Ergebnisse geliefert. Jedoch zeigt das Verfahren beim Datensatz CIFAR10 mangelhafte Leistungen. Wie in den vorherigen Kapiteln dargestellt, hängt die Performanz des DECs stark vom Zustand des trainierten Autoencoders ab. Falls der Autoencoder gute initiale Zentroide liefert, folgt eine erfolgreiche Verfeinerung des eingebetteten Raums. Auf lineare Netze ist der Umgang mit den Farbbild-Datensätzen, wie CIFAR10, schwer (siehe Abbildung 6.5). Aus diesem Grund kann der *Stacked Convolutional Autoencoder* [46] angewendet werden. Viele der erweiterten Verfahren des DECs verwenden durchaus den Convolutional Autoencoder, um die Parameter-Initialisierungsphase zu realisieren [40] [25]. Der Convolutional Autoencoder führt zusätzlich die Convolutional-Schichten ein. Die Convolutional-Schichten versuchen die räumlichen Informationen der eingegebenen Bilddaten beizubehalten, indem *Convolution* Operationen angewendet werden. Eine Convolution dient zur Merkmalsextrahierung des Bildes anhand eines gegebenen Filters und reduziert gleichzeitig die Größe der Eingabe (durch Erzeugung der *Feature Maps*). Der *Stride* ist ein Hyperparameter und dient zur Bestimmung der Schrittgröße für die Anwendung der Operation. Das bedeutet, für eine Schrittgröße von 2 werden die Filter jeweils um 2 Pixel verschoben. Die *Pooling* Schicht ist eine weitere Schicht, die zur Reduzierung der Größe dient. Eine bekannte Pooling-Schicht ist die Max-Pooling, in der das größte Element aus dem Filter-Bereich ausgewählt wird.

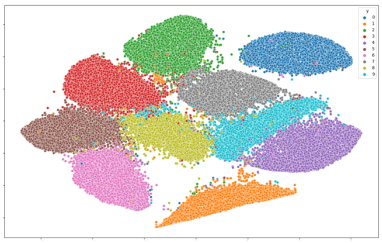
Zwar erreicht der Convolutional Autoencoder grundsätzlich einen geringeren Verlust im Vergleich zum linearen Autoencoder, jedoch liegt das Interesse hauptsächlich auf die Qualität des trainierten, eingebetteten Raums in Bezug auf die Clusteringanalyse, sodass eine nähere Analyse notwendig ist. In der Abbildung B.4 werden zwei Varianten des Stacked Convolutional Autoencoders vorgestellt. Die linke Abbildung stellt einen Convolutional Autoencoder mit der Pooling-Schicht dar, wobei der Autoencoder zunächst aus drei wiederholenden Teilen aus einer Convolutional Schicht mit Filter der Größe 4 und ReLu als Aktivierungsfunktion, und einer Max-Pool Schicht mit Filter der Größe 2 besteht. Nach den drei Convolutional Schichten folgt schließlich eine Fully Connected Schicht. Die rechte Abbildung stellt einen Convolutional Autoencoder ohne die Pooling-Schicht dar. Der Autoencoder besteht somit nur aus drei Convolutional Schichten mit der Filtergröße von 4 und ReLu als Aktivierungsfunktion und schließlich einer Fully Connected Schicht. Beide Antoencoder wurden anhand der Minimierung des Rekonstruktionsfehler (MSE und ADAM[10]) für 100 Epochen auf dem Datensatz MNIST trainiert. Es wurde festgestellt, dass der Autoencoder mit Pooling-Schichten einen deutlich niedrigeren Rekonstruktionsfehler (von ungefähr 0.025) erzeugt. Die bildlichen Darstellungen der beiden Autoencoder in der Abbildung B.4 zeigen jedoch deutlich andere Ergebnisse. Zwar konvergiert der eingebettete Raum vom Autoencoder mit Pooling-Schichten gegen ein bestimmtes Muster, jedoch

[10] mit 0.001 Lernrate. `https://pytorch.org/docs/stable/generated/torch.optim.Adam.html`

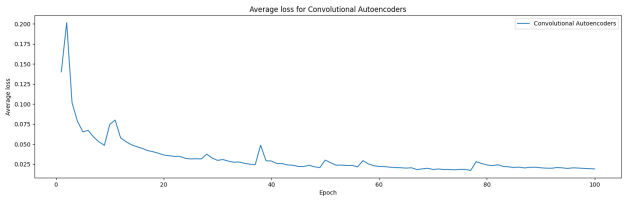
ist der Raum für die Clusteringanalyse ungeeignet. Andererseits erstellt der Autoencoder mit Strides einen deutlich angenehmeren Raum für eine erfolgreiche Clusteringanalyse. Für den Datensatz CIFAR10 zeigen jedoch beide Konfigurationen des Convolutional Autoencoders wenig Erfolg, sodass die Hyperparameter gegebenfalls erneut angepasst werden müssen. Nichtsdestotrotz ist das selbe Verhalten wie beim Datensatz MNIST zu erkennen. Zwar erreicht der Convolutional Autoencoder mit Pooling Schichten einen besseren Rekonstruktionsfehler von ungefähr 0.01 (im Vergleich zu 0.02 beim Convolutional Autoencoder ohne Pooling Schicht), jedoch ist im eingebetteten Raum ein ähnliches Muster wie beim MNIST zu erkennen (Abbildungen zum CIFAR10 befinden sich im Anhang B.3). Zusammengefasst lässt sich feststellen, dass die Datenpunkte beim Convolutional Autoencoder mit Pooling-Schichten „gezwungen“ werden, in einem großen Cluster/Kreis zu bleiben. Ohne Pooling-Schichten scheinen die Daten „freier“ bewegen zu können, sodass einzelne unabhängige Cluster gebildet werden können.



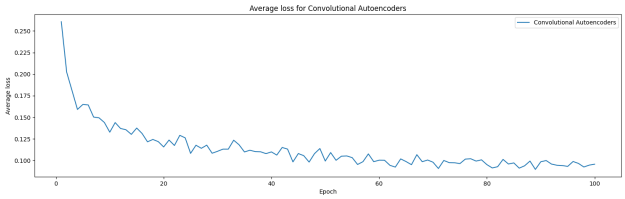
(a) SCA mit Pooling-Schichten



(b) SCA mit Strides



(c) Verlauf des durchschnittlichen Verlustes vom SCA mit Pooling-Schichten



(d) Verlauf des durchschnittlichen Verlustes vom SCA mit Strides

Abbildung 6.4: Illustrierung des eingebetteten Raums nach 100 Epochen anhand des TSNE Verfahrens und Darstellung des durschnittlichen Verlustes der jeweiligen Epoche für zwei Stacked Convolutional Autoencoder (SCA) [**Datensatz MNIST**]

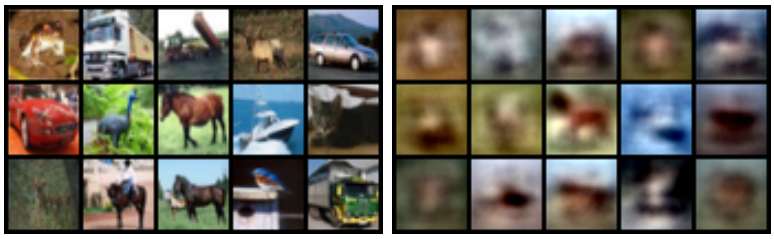


Abbildung 6.5: Rekonstruktionserfolg des DECs mit einem linearen Stacked Autoencoder auf dem **Datensatz CIFAR10**

6.3.5 Diskussion über die Notwendigkeit der gemeinsamen Optimierung

In diesem Abschnitt geht es um die Fragestellung nach der Notwendigkeit der von den Autoren vorgeschlagenen gemeinsamen Optimierung in der Parameter-Optimierungsphase des DECs. Die paraelle Optimierung der eingebetteten Beobachtungen und der einzelnen Clusterzentroide aus dem Kapitel (4.2.2) dienen zur Verfeinerung der einzelnen Cluster. Es stellt sich die Frage, wie viel Einfluss die Optimierung der einzelnen Clusterzentroide anhand der KL-Divergenz auf die tatsächliche neu-Positionierung und folglich auf das Clutering- und Zusammenfassungsergebnis hat. Anhand der Diagramme der Abbildung 6.6 ist zu erkennen, dass die gemeinsame Optimierung keine entscheidende Rolle für die Performanz des DECs auf dem Datensatz MNIST spielt. Jedoch zeigen die Diagramme der Abbildung 6.7 für den Datensatz CIFAR10 durchaus unterschiedliches Verhalten bei den Accuracies. Hierbei wird gefragt, wie das Verhältnis zwischen zwei Optimierungen gewesen wäre, falls ein passender Autoencoder für den Datensatz CIFAR10 gewählt wurde.

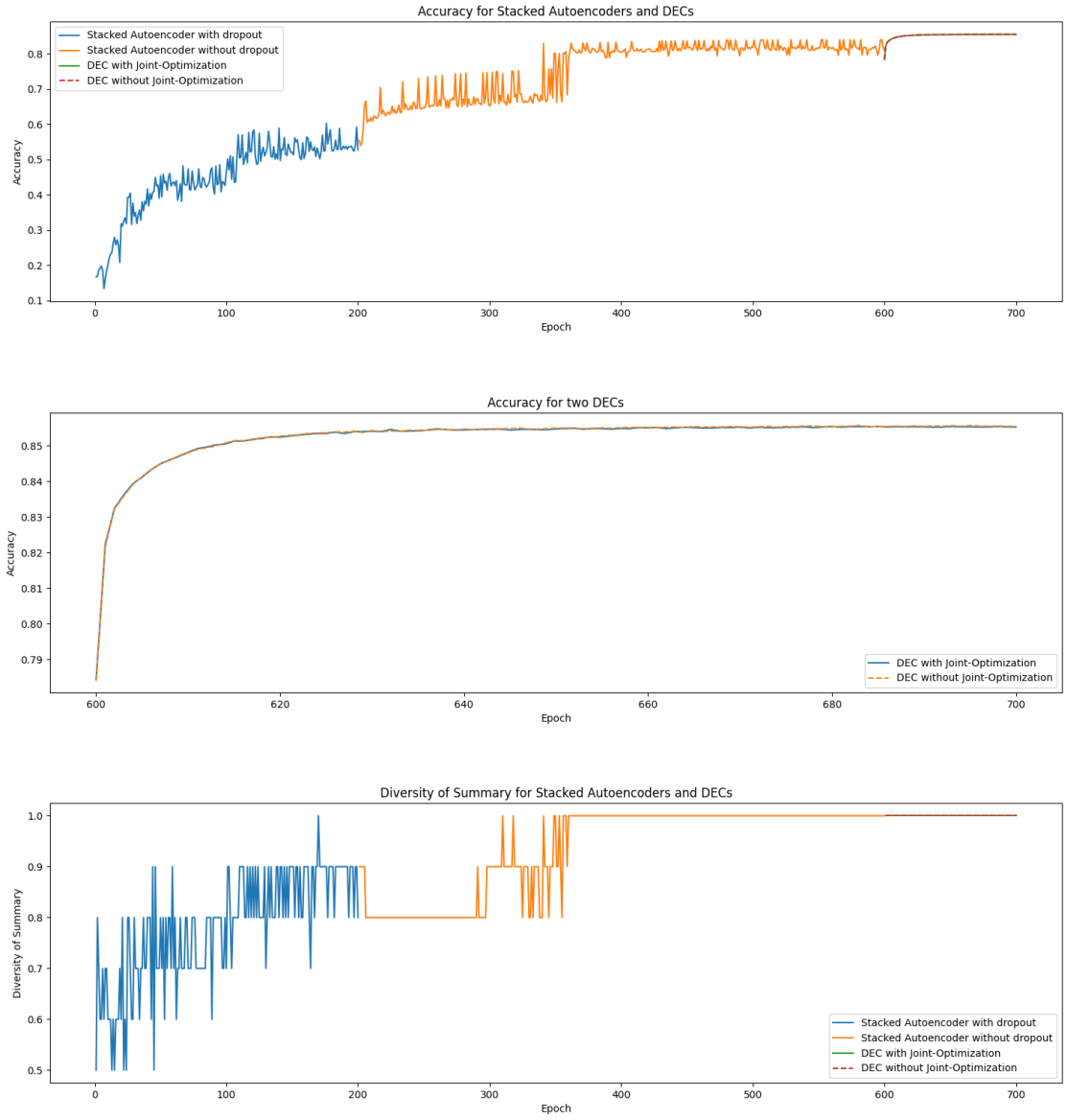


Abbildung 6.6: Darstellung der ACC (die ersten zwei obersten Diagramme) und SDM (das unterste Diagramm) im Verlauf der Parameter-Initialisierungsphase, bzw. der Trainingsphase des Stacked Autoencoders mit und ohne Dropout, und der Parameter-Optimierungsphase des DECs. DEC with Joint-Optimization stellt die ursprüngliche, gemeinsame Optimierung der eingebetteten Beobachtungen und der initialen Zentroide von Xie et al[60] dar. DEC without Joint-Optimization stellt die Optimierung nur nach der eingebetteten Beobachtungen und ohne die Aktualisierung der initialen Zentroide dar [**MNIST Datensatz**]

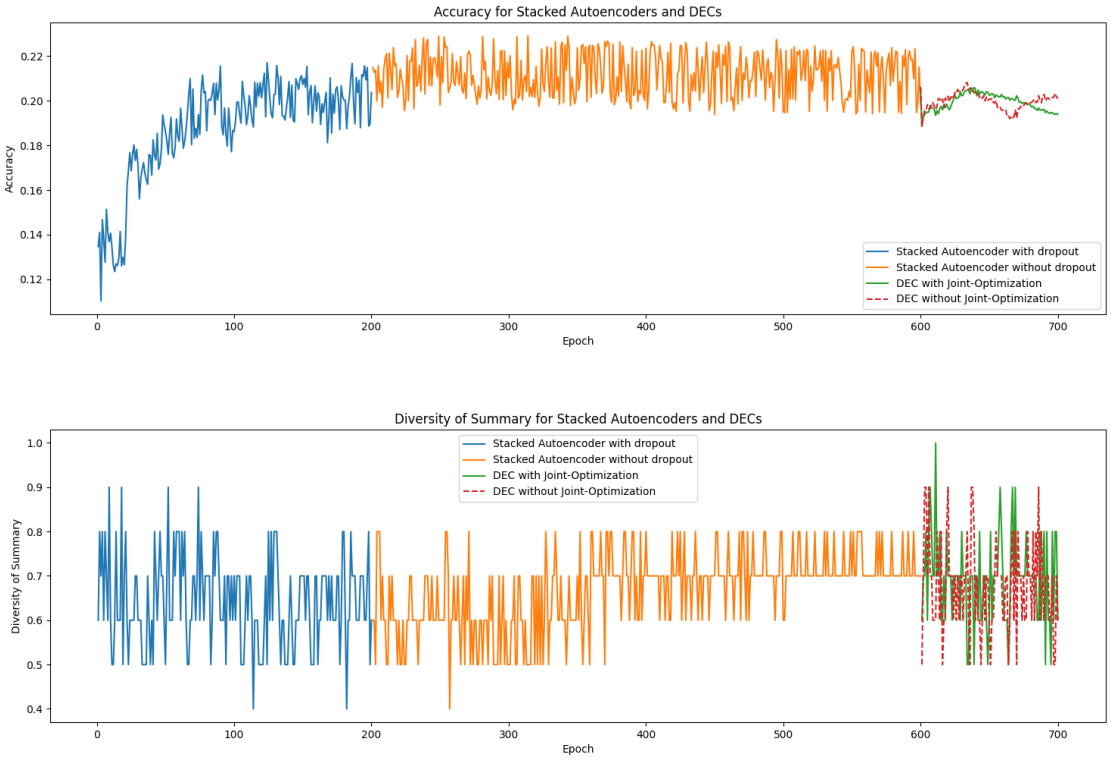


Abbildung 6.7: Darstellung der ACC (das obere Diagramm) und SDM (das untere Diagramm) im Verlauf der Parameter-Initialisierungsphase, bzw der Trainingsphase des Stacked Autoencoders mit und ohne Dropout, und der Parameter-Optimierungsphase des DECs. DEC with Joint-Optimization stellt die ursprüngliche, gemeinsame Optimierung der eingebetteten Beobachtungen und der initialen Zentroide von Xie et al[60] dar. DEC without Joint-Optimization stellt die Optimierung nur nach der eingebetteten Beobachtungen und ohne die Aktualisierung der initialen Zentroide dar [**CIFAR10 Datensatz**]

6.4 Experiment II: Performanz von DSA

In diesem Abschnitt wird der Deep Submodulare Autoencoder eingeführt, um eine möglichst effiziente Zusammenfassung zu erzielen. Zunächst wird eine mögliche Implementierungsidee des Konzepts des DSAs vom Kapitel 5 vorgestellt. Anschließened wird der DSA auf dem Datensatz MNIST angewendet und die resultierenden Ergebnisse evaluiert. Schließlich wird über Auffälligkeiten des DSAs anhand der untersuchten Ergebnisse und über eine mögliche Verbesserung der vorgestellten Implementierung diskutiert.

6.4.1 Implementierung des DSAs

In dieser Arbeit verwendet der Submodulare Autoencoder ebenfalls einen linearen Stacked Autoencoder. Die Netzwerk-Dimension des DSAs beträgt $d-500-500-2000-10$, wobei d die Dimensionalität der jeweiligen Datensätze darstellt. Im Gegensatz zum DEC wird der DSA ohne die Dropout Schichten konfiguriert. Die Dropout Schichten dienen insbesondere zur Regularisierung (engl. *Regularization*) des Netzes. Da das Verfahren in dieser Arbeit nicht nach einer guten Vorhersage strebt und außerdem ohne die Dropout Schichten auch ein guter Rekonstruktionsfehler erreicht werden kann, werden diese Schichten nicht mehr in Betracht gezogen. Des Weiteren ist die initiale Konfiguration des DSAs identisch mit der von DEC. Anschließend folgen die wesentlichen Unterschiede des DSAs zum DEC. Der erste Unterschied ist die 2-Phasen Teilung. Eine Besonderheit des DSAs ist die Vereinigung der Parameter-Initialisierungsphase und der Parameter-Optimierungsphase. Folglich werden die Rekonstruktionsfehler und die KL-Divergenz während des gesamten Prozesses gemeinsam betrachtet. Diese Idee entspricht der Gleichung (5.11) aus dem Kapitel 5.
Der zweite Unterschied ist die Einführung der submodularen Maximierung zur Auswahl der Zentroide am Ende jeder Epoche (siehe Gleichung (5.12) des Kapitels 5). In dieser Arbeit wird der gierige Algorithmus aus dem Kapitel 2.3.1 zum Lösen der submodularen Maximierung verwendet. Außerdem wird die logarithmische Determinante des RBF-Kernels aus der Informative Vector Machine vom Kapitel 2.2.1 als die zu maximierende, submodulare Funktion ausgewählt. Die Skalierungskonstante σ der Funktion wird in den folgenden Experimenten auf 1 gesetzt. Die Python und C++ Implementierung des gierigen Algorithmus und der logarithmischen Determinante sind unter dem folgenden Link[11] frei verfügbar. Insgesamt wird der DSA für 700 Epochen mit der kombinierten Verlustfunktion aus MSE und KL-Divergenz anhand des stochastischen Mini-Batch Gradientenverfahrens der Größe 256 trainiert. Hierbei wird ebenfalls die Lernrate am Anfang auf 0.1 mit einem Momentum[12] von 0.9 gesetzt und alle 300 Epochen durch 10 geteilt. Zusätzlich sind bei der Implementierung zwei folgende Aspekte besonders zu beachten.

[11] https://github.com/sbuschjaeger/SubmodularStreamingMaximization
[12] https://pytorch.org/docs/stable/generated/torch.optim.SGD.html

Parameterauswahl

Eine große Herausforderung der Implementierung des DSAs ist, die folgenden Parameter passend auszuwählen: die Konstante α des RBF-Kernels (2.8), wobei $\alpha = 2l^2$ gilt, und der Regulierungsfaktor λ aus der Gleichung 5.11. Für die kommenden Experimente werden für den Regulierungsfaktor λ die Konstanten 0.05 und 0.1, und für den Parameter α die Werte 3.1, 5.5, 8.0 und 10.0 betrachtet.

Einführung der Zufallsprozesse

Das Verfahren DEC lässt den K-Means-Algorithmus mit zufällig ausgewählten Zentroiden 20 mal unabhängig voneinander laufen und nimmt davon die Clusterzentroide, aus denen der kleinste Inertia erfolgt. Folgende Auswahlidee kann ebenfalls für die Erstellung einer möglichst erfolgreichen Zusammenfassung angewendet werden. Das Ziel ist am Ende jeder Epoche mehrere unterschiedliche Zusammenfassungen zu erstellen und aus diesen die jenige mit dem höchsten Score auszuwählen. Fraglich ist jedoch, wie die Zufälligkeit in die submodulare Maximierung einzuführen ist, sodass tatsächlich unterschiedliche Zusammenfassungen erstellt werden. Die logarithmische Determinante der Summe zwischen der Einheitsmatrix und einer leeren Kernelmatrix beträgt 0. Folglich sind alle Elemente eines Datensatzes bei einer leeren Zusammenfassung gleichviel Wert. In dieser Arbeit wählt der gierige Algorithmus immer das erstes Element eines Datensatzes, falls alle Elemente den gleichen marginalen Zuwachs erzeugen. Somit besteht die Zufälligkeit darin, für jede neue Berechnung der Zusammenfassung den Datensatz so zu mischen[13], dass sich am Anfang der Zusammenfassung immer ein neues Element befindet. Da das erste Element für die Auswahl weiterer Elemente ausschlaggebend ist, wird damit versucht, unterschiedliche Zusammenfassungen zu erstellen. Zusammengefasst besteht ein „Zufallsprozess“ darin, den gesamten Datensatz einmal zu mischen und anschließend den gierigen Algorithmus auszuführen. Zunächst werden 20 Zufallsprozesse[14] am Ende jeder Epoche betrachtet. Aus den 20 unterschiedlichen Zusammenfassungen wird diejenige mit dem höchsten Score ausgewählt und damit weitergearbeitet.

[13] Die Mischung wird anhand dieser Funktion ausgeführt: `https://scikit-learn.org/stable/modules/generated/sklearn.utils.shuffle.html`

[14] Die Durchführung von 20 Zufallsprozessen am Ende jeder Epoche bedeutet: die Erstellung von 20 unterschiedlichen Zusammenfassungen am Ende jeder Epoche nach dem vorgestellten Zufallsprinzip

6.4.2 Resultat auf dem Datensatz MNIST

Für den Kernel-Parameter $\alpha = 3.1$ mit dem Regulierungsfaktor $\lambda = 0.05$ und den Kernel-Parameter $\alpha = 5.5$ mit dem Regulierungsfaktor $\lambda = 0.1$ liefert DSA den höchsten SDM-Accuracy, nämlich 60%.

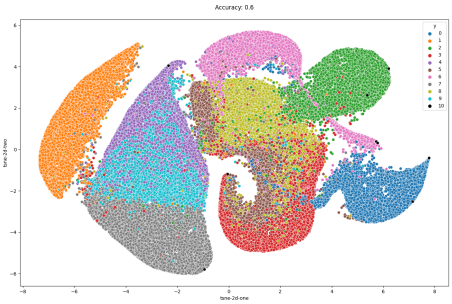
		λ	
		0.05	0.1
σ des Kernels	3.1	60%	40%
	5.5	50%	60%
	8.0	50%	40%
	10.0	50%	30%

Tabelle 6.2: Die SDM-Accuracies vom DSA bezüglich des Regulierungsfaktors λ und des Kernel-Parameters α

Diskussion über das Resultat

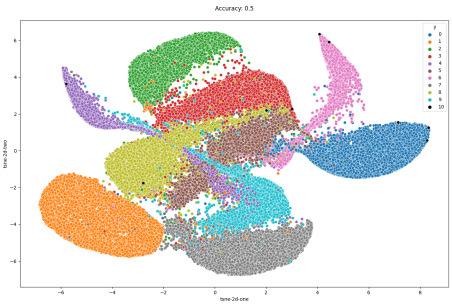
In der Tabelle 6.2 sind die SDM-Accuracies in Bezug auf dem Kernel-Parameter α und dem Regulierungsfaktor λ zu sehen. Durchschnittlich liefert DSA mit einem kleineren λ einen besseren SDM-Accuracy, trotzdem führt er nur zu einer mangelhaften Performanz. Die Abbildung 6.8 zeigt den Verlauf der zwei DSAs, wobei sie sich nur in dem Regulierungsfaktor λ unterscheiden. Beim Regulierungsfaktor $\lambda = 0.1$ ist eine deutlich stärkere „Anziehung“ der Beobachtungen zu den Elementen der Zusammenfassung zu sehen. Nach 700 Epochen sind die Auswirkungen der beiden Regulierungsfaktoren eindeutig zu erkennen. Mit einem höheren λ konvergieren die Beobachtungen stärker gegen eine Linie. Dieses Verhalten ist auch beim DEC in der Optimierungsphase zu erkennen. Insbesondere stellt sich die Frage, wie sich das Verhältnis zwischen dem Rekonstruktionsfehler und der KL-Divergenz während des Prozesses aussieht. Notfalls müsste ein weiterer Regulierungsparameter in die Verlustfunktion eingeführt werden, welcher sich auch im Verlauf des Prozesses ändert.

Obwohl die Notwendigkeit zur weiteren Anpassung des Regulierungsfaktors besteht, hat die ausgewählte Zusammenfassung während des DSA-Verfahrens eine größere Verantwortung für den Misserfolg. Die TSNE-Abbildungen und die Diagrammen aus 6.8 zeigen, dass in keiner Epoche eine Zusammenfassung aus den Elementen aller Klassen erstellt wurde. Im nächsten Abschnitt werden die erstellten Zusammenfassungen während des DSAs genauer untersucht und auf die einzelnen Implementierungen eingegangen.

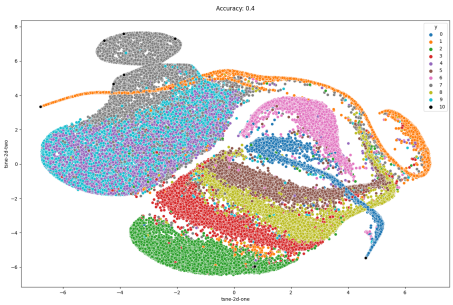


(a) Nach Epoche 50 für $\alpha = 3.1$, $\lambda = \mathbf{0.05}$
SDM = 60%

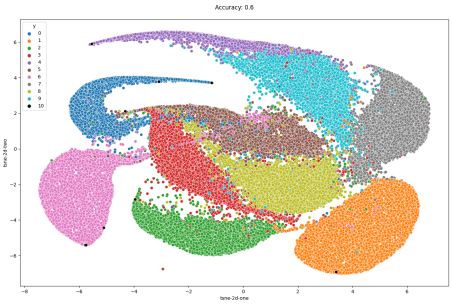
(b) Nach Epoche 50 für $\alpha = 3.1$, $\lambda = \mathbf{0.1}$
SDM = 50%



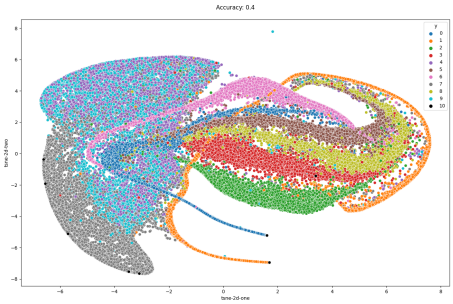
(c) Nach Epoche 300 für $\alpha = 3.1$, $\lambda = \mathbf{0.05}$
SDM = 50%



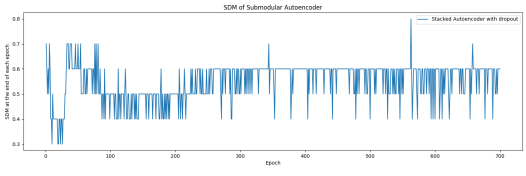
(d) Nach Epoche 300 für $\alpha = 3.1$, $\lambda = \mathbf{0.1}$
SDM = 40%



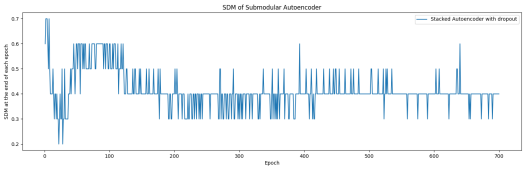
(e) Nach Epoche 700 für $\alpha = 3.1$, $\lambda = \mathbf{0.05}$
SDM = 60%



(f) Nach Epoche 700 für $\alpha = 3.1$, $\lambda = \mathbf{0.1}$
SDM = 40%



(g) SDM-Accuracies des DSA mit $\alpha = 3.1$, $\lambda = \mathbf{0.05}$



(h) SDM-Accuracies des DSA mit $\alpha = 3.1$, $\lambda = \mathbf{0.1}$

Abbildung 6.8: Vergleich der zwei DSA Prozesse mit unterschiedlichen λ. Die **linke Spalte** stellt den DSA-Verlauf mit $\alpha = 3.1$ und $\lambda = \mathbf{0.05}$ und die **rechte Spalte** stellt den DSA-Verlauf mit $\alpha = 3.1$ und $\lambda = \mathbf{0.1}$ dar (weitere Abbildungen in C.1, C.2)

6.4.3 Diskussion: Herausforderungen bei der Implementierung

DSA weist keine erwartete Leistung bezüglich des Zusammenfassungsproblems auf. In diesem Abschnitt werden Vermutungen darüber angestellt, was genau an der Implementation gescheitert haben könnte und über mögliche Verbesserungen diskutiert.

Anzahl der Zufallsprozesse

Zunächst könnte die schlechte Leistung an der niedrigen Anzahl der ausgeführten Zufallsprozesse liegen. Ein naiver Verbesserungsvorschlag ist, am Ende jeder Epoche eine höhere Anzahl an Zusammenfassungen nach dem vorgestellten Zufallsprinzip zu erstellen, sodass die Wahrscheinlichkeit des Vorkommens einer optimalen Zusammenfassung erhöht wird. Die Frage ist jetzt, was die minimale Anzahl der notwendigen Zufallsprozesse ist. Folglich wird ein kleines Experiment eingeführt. Betrachtet werden 20, 100 und 500 Zufallsprozesse. Da das Interesse ebenfalls auf die optimale Auswahl des Kernel-Parameters α liegt, werden für alle 20, 100 und 500 Zufallsprozesse auch die verschiedenen α Parameter betrachtet. Eine Besonderheit bei diesem Experiment ist, dass alle Berechnungen aus dem selben eingebetteten Raum erfolgen. Dieser Raum entspricht dem eingebetteten Raum nach der Initialisierungsphase des DECs auf dem Datensatz MNIST. Außerdem achtet das Experiment auf die Abhängigkeit zwischen den jeweiligen Zufallsprozessen, sodass die 20 unterschiedlichen Zusammenfassungen eine Teilmenge von 100 unterschiedlichen Zusammenfassungen sind und diese 100 unterschiedliche Zusammenfassungen eine Teilmenge von 500 unterschiedlichen Zusammenfassungen sind. Daher werden in diesem Experiment 500 Zufallsprozesse auf dem eingebetteten Raum durchgeführt und 3 Mengen erstellt, wobei die erste Menge die ersten 20 Zusammenfassungen, die zweite Menge die ersten 100 Zusammenfassungen und die dritte Menge alle Zusammenfassungen enthält. Schließlich wird bei den drei Mengen jeweils die Zusammenfassung mit dem höchsten Score ausgesucht und der SDM-Accuracy berechnet. Die berechneten SDM-Accuracies werden in der Tabelle 6.3 dargestellt.

		Zufallsprozesse		
		20	100	500
σ des Kernels	3.1	70%	60%	80%
	5.5	70%	70%	70%
	8.0	70%	60%	70%
	10.0	70%	70%	60%

Tabelle 6.3: Die SDM-Accuracies

In der Tabelle 6.3 ist zu erkennen, dass eine Anzahl von 500 Zufallsprozessen mit dem Kernel-Parameter $\alpha = 3.1$ den höchsten SDM-Accuracy liefert.

Unerwartetes Verhalten der Submodularität

Die Tabelle 6.3 zeigt nicht nur die notwendige Anzahl der Zufallsprozesse, sondern auch ein unerwartetes Verhalten bezüglich der Submodularität. Beispielsweise stellt die Menge der 20 Zusammenfassungen aus 20 Zufallsprozessen eine Teilmenge der 100 Zusammenfassungen aus den 100 Zufallsprozessen dar (mit $\alpha = 3.1$). Falls keine bessere Zusammenfassung während der 80 verbleibenden Zufallsprozesse erzeugt wird, entspreche die beste Zusammenfassung aus den vorherigen 20 Zufallsprozessen der besten für die 100 Zufallsprozesse. Aus diesem Experiment wird zeilenweise entweder ein Wachstum oder eine Stagnation der SDM-Accuracies erwartet. Die SDM-Accuracies aus der Tabelle zeigen jedoch nicht das erwartete Verhalten. Aus diesem Grund stellt sich die Frage, *ob tatsächlich der höchste Score zu einer optimalen Zusammenfassung führt*. Um weiteres Verständnis über die Beziehung zwischen dem Score und der Zusammenfassung zu erhalten, werden zwei Abbildungen eingeführt. Die Abbildungen aus 6.9 zeigen wie die Elemente der Zusammenfassung im eingebetteten Raum abgebildet sind. In der linken Abbildung handelt es sich um die beste Zusammenfassung[15] aus der 20 Zufallsprozesse und in der rechten Abbildung um die beste Zusammenfassung aus der 100 Zufallsprozesse, wobei die 20 Zusammenfassungen eine Teilmenge der 100 Zusammenfassungen darstellen. Außerdem wurden alle Zufallsprozesse mit $\alpha = 3.1$ durchgeführt. In der rechten Abbildung ist eindeutig zu sehen, dass im gelben Cluster viele Beobachtungen in die Zusammenfassung hinzugefügt wurden. Insbesondere zeigen diese Beobachtungen einen geringen Abstand zueinander. Dahingegen zeigen die ausgewählten Beobachtungen aus der linken Abbildung eine bessere Verteilung, somit auch einen höheren SDM-Accuracy. Obwohl die Zusammenfassung in der rechten Abbildung eine schlechtere Verteilung aufweist, ist der Score dieser Zusammenfassung höher als die von der linken Abbildung. Die Frage, *wieso die schlechter verteilte Zusammenfassung einen höheren Score zeigt*, können folgende Vermutungen aufklären:

- **Grund 1:** In der Tat führt der höchste Score nicht immer zu einer optimalen Zusammenfassung. Folgendes Szenario kann bei der rechten Abbildung aus 6.9 aufgetreten sein. Das erste, ausgewählte Element der Zusammenfassung ist eine Beobachtung aus dem gelben Cluster. Demnächst wurden andere Beobachtungen möglichst weit entfernt von dem ersten Element ausgewählt (also nicht aus dem gelben Cluster). Statt danach weitere Elemente, die ebenfalls eine weite Entfernung von dem ersten Element aufweisen, zu wählen, wurden Beobachtungen in der Nähe vom ersten Element (im gelben Cluster) genommen, weil diese insgesamt besonders große Distanzen zu allen anderen Elementen der Zusammenfassung haben. Das bedeutet, die Distanzen der ausgewählten Elemente aus dem gelben Cluster sind so weit entfernt von den restlichen Zusammenfassungenselementen, dass diese Auswahl trotz der kurzen Entfernung zum ersten Element, insgesamt zu einem sehr hohen Score führt. In einem

[15] höchster zugewiesener Score

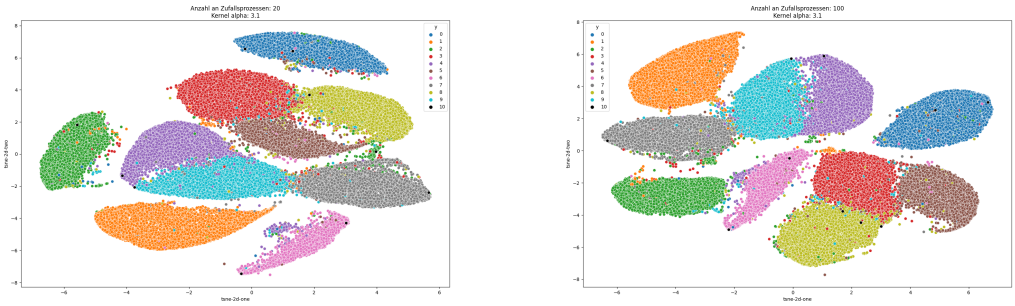


Abbildung 6.9: Illustrierung der selben eingebetteten Räume anhand des TSNE-Verfahrens. Die Zusammenfassung der linken Abbildung entspricht der mit dem höchsten Score aus den 20 Zufallsprozessen mit $\alpha = 3.1$. Die Zusammenfassung der rechten Abbildung entspricht der mit dem höchsten Score aus den 500 Zufallsprozessen mit $\alpha = 3.1$. Hierbei entsprechen die Zusammenfassungen aus 20 Zufallsprozessen eine Teilmenge von den Zusammenfassungen aus 100 Zufallsprozessen (Weitere Abbildungen in C.3)

eingebetteten Raum, in dem ein Cluster sich besonders weit entfernt von allen anderen Clustern befindet (wobei die anderen Cluster deutlich geringere Abstände zueinander haben), kann eine Distanz-basierte, submodulare Funktion, die beispielsweise die Kernel-Funktion für die Ähnlichkeitsberechnung verwendet, deutlich schlechtere Leistung zeigen. Folglich muss das Verhalten weiterer submodularer Funktionen ebenfalls genauer betrachtet werden.

- **Grund 2:** Oder tatsächlich handelt es sich um eine optimale Zusammenfassung. Das bedeutet, dass die ausgewählten Elemente im gelben Cluster in der rechten Abbildung 6.9 alle weit voneinander liegen. Zwar gelingt es dem TSNE-Verfahren die Nachbarschaftsbeziehungen von der ursprünglichen Dimension gut zu erfassen, jedoch zeigt es bei der Erhaltung der Distanzbeziehungen keinen Erfolg. Folglich illustriert das TSNE-Verfahren den selben Cluster im identischen, eingebetteten Raum mit unterschiedlichen Größen und Formen (siehe Abbildungen 6.9). Die TSNE-Abbildungen scheitern in diesem Fall eine gute Interpretation zu liefern. Hierfür muss die Auswirkung der Submodulariät auf einem Raum mit unterschiedlichen Formen und Größen der Cluster untersucht werden.

Kapitel 7

Fazit und Ausblick

In dieser Arbeit wurde das Zusammenfassungsproblem anhand der Deep Learning Ansätze untersucht. DEC stellt sich insgesamt als ein gutes Verfahren für die Clusteringanalyse der hoch dimensionalen Datensätze dar. Für das Zusammenfassungsproblem hat DEC ebenfalls ein gutes Verhalten gezeigt und die nächsten Beobachtungen der Zentroide waren eine gute Auswahl für eine optimale Zusammenfassung. Dennoch hat DEC einige Schwäche und Einschränkungen aufgewiesen. Es hat sich als sehr abhängig von der Performanz des Autoencoders und des Clusteringverfahrens erwiesen. Dementsprechend kann DEC leicht zu einem Fehlverhalten führen. Im Gegensatz zu DEC, weist DSA ein gutes, theoretisches Verhalten auf das Zusammenfassungsproblem auf. Die Schwierigkeit in dieser Arbeit hat sich aus der genauen Implementierung des DSAs ergeben. Eine besondere Herausforderung war das unerwartete Verhalten der logarithmischen Determinante der RBF-Kernel Funktion. Hierbei wurde mit dem gierigen Algorithmus nicht die erwünschte Zusammenfassung erzeugt. Folglich müssen drei Aspekte genauer untersucht werden. Als erstes muss das Verfahren anhand weiterer submodularer Funktionen untersucht werden. Insbesondere müssen submodulare Funktionen ausgewählt werden, welche die Ähnlichkeitsverhältnisse nicht anhand der Distanzen berechnen. Als nächstes muss der eingebettete Raum des DSAs genauer betrachtet werden. Hierbei stellt sich die Frage, ob der eingebettete Raum einen geeigneten Raum für die Anwendung der submodularen Funktion darstellt. Außerdem kann das unerwartete Verhalten an der Dimensionalität gelegt haben, sodass die Arbeit mit der submodularen Funktion in dem eingebetteten Raum der Dimension 10 vom Fluch der Dimensionalität betroffen war.

Anhang A

Anhang I

A.0.1 Weitere Optimierungsalgorithmen

Die Erstellung eines Algorithmus, welches eine bessere Güte als der gierige Algorithmus garantiert, ist selbst ein NP-schweres Problem [19] [21]. Der gierige Algorithmus zeigt jedoch eine eindeutige Schwäche in der Laufzeit, falls die Kosten der Funktionsauswertung oder die Größe der Zusammenfassung zu groß wird. Nichtsdestotrotz bildet der gierige Algorithmus die Grundlage für viele der folgenden Algorithmen. **Lazy Greedy** [47] bietet beispielsweise eine gute Implementierungsmöglichkeit an. Statt den marginalen Zuwachs für jedes Element in V zu berechnen, wird eine absteigend sortierte Prioritätenliste für jedes Element in Bezug auf den marginalen Zuwachs verwaltet und aktualisiert bis der Kopf weiterhin das größte Element bleibt. Folgich wird eine erneute Berechnung der uninteressanten Elemente verhindert [48] [55]. Obwohl die exakte Laufzeit in Bezug auf die Funktionsauswertungen unbekannt ist, führt der Ansatz in der Praxis zur Beschleunigung um Größenordnungen [48]. Dennoch bleibt die Worst-Case Laufzeit $O(k \cdot |V|)$, weshalb nach der Unabhängigkeit von der Größe k gestrebt wird. **Stochastic Greedy** schafft eine lineare Lautzeit $O(|V| \log(1/\exp(1)))$ von Funktionsauswertungen für monotone, submodulare Funktionen, indem bei jeder Iteration nur eine zufällig ausgewählte Teilmenge der Größe $((|V|/k) \log(1/\epsilon))$ betrachtet wird. Hierbei erfolgt die Zufälligkeit unter einer Gleichverteilung. Darauffolgend erreicht der Algorithmus eine Approximationsgarantie $\alpha_{stochgreedy} \geq (1 - (1/exp(1)) - \epsilon)$ in Erwartung, sodass $E[f(S_{stochgreedy})] \geq (1 - (1/exp(1)) - \epsilon) f(S^*)$ gilt [19] [48].

Eine weitere mögliche Umsetzung der linearen Funktionsaufrufe ist die Verwendung des Datenstromalgorithmus. Hierbei werden die Daten als ein Datenstrom angenommen, wodurch alle Elemente nur einmal betrachtet werden müssen und die Laufzeit folglich auf $O(|V|)$ reduziert wird. Eine einfache Erweiterung des gierigen Algorithmus ist der **Stream Greedy** [23]. Zunächst werden die ersten k Elemente in die Zusammenfassung eingefügt. Für alle weiteren Elemente wird überprüft, ob ein Austausch des neuen Elements mit ei-

nem vorhandenen Element zur Erhöhung des Scores $f(S_{streamgreedy})$ führt und falls die Erhöhung stattfindet, erfolgt der Austausch. Stream Greedy liefert eine Approximationsgarantie $\alpha_{streamgreedy} \geq ((1/2) - \epsilon)$, falls mehrere Durchläufe über die Daten erlaubt sind. Eine verbesserte Erweiterung des Stream Greedys ist der **SieveStreaming-Algorithmus** [2]. Zunächst wird ein Schwellenwert (egl. $Threshhold$) eingeführt, sodass das Einfügen in die Zusammenfassung während der Laufzeit nur ab einem bestimmten Zuwachs des Scores erfolgt. Vorallem findet kein Austausch mehr statt und es handelt sich um einen *One Pass Algorithmus* [2]. Die Idee des Schwellenwertes lässt sich wie folgt formalisieren. Das Element x_i wird in die Menge S_p der Größe M hinzugefügt, falls für $OPT \geq p \geq \alpha\, OPT$ gilt: $\Delta_f(x_i|S_p) \geq (f(S_p)/2 - f(S_p))/(M - |S_p|)$. Hierbei stellt p den approximierten Wert des optimalen Scores dar. Offen steht noch die Auswahl eines guten, approximierten Werts p. Jedoch erfordert die folgende Approximation die Kenntnis über den Wert OPT. Aus der Submodularität folgt: $m \leq OPT \leq k \cdot m$, wobei m den maximalen Score einer einelementigen Teilmenge $m = max_{x \in V} f(x)$ darstellt. Folglich lassen sich mehrere Schätzungen des OPT aufstellen. Aus der Menge $O = \{(1+\epsilon)^i | i \in \mathbb{Z}, m \leq (1+\epsilon)^i \leq k \cdot m\}$ nimmt SieveStreaming $(log(k)/\epsilon)$ verschiedene Schätzungen zum Ausprobieren, sodass mindestens eine Schätzung $p \in O$ eine Approximationsgarantie von $(1-\epsilon)$ liefert: $(1-\epsilon)OPT \leq p \leq OPT$. Schließlich laufen insgesamt $(log(k)/\epsilon)$ paraelle Durchführungen des Algorithmus für $(log(M)/\epsilon)$ unterschiedliche Schwellenwerte. Da es sich um einen Datenstromalgorithmus handelt, muss der Wert m auch während der Ausführung berechnet werden. SieveStreaming aktualisiert den Wert m, falls das eingetroffene Element e_i einen besseren Score $f(e_i)$ liefert: $m = \max(m, f(e_i))$. Bei jeder Aktualisierung des Werts m, berechnet der Algorithmus die Menge O neu und löscht alle Schwellenwerte außerhalb der Menge, wobei die Grenze der Menge verdoppelt wird. Schließlich ergeben sich aus dem gesamten Algorithmus maximal $(log(k)/\epsilon)$ verschiedene Zusammenfassungen und die Zusammenfassung, welche den größten Score zurückliefert, wird ausgegeben. Alles in allem ergibt sich aus dem Algorithmus eine Speicherkapazität von $O(k(log(k)/\epsilon))$, eine Laufzeit $O(|V| \cdot (log(k)/\epsilon))$ von Funktionsaufrufen und eine Approximationsgarantie $\alpha_{sievesstreaming} \geq ((1/2) - \epsilon)$.

Anhang B

Anhang II

B.1 Abbildungen zum Experiment I

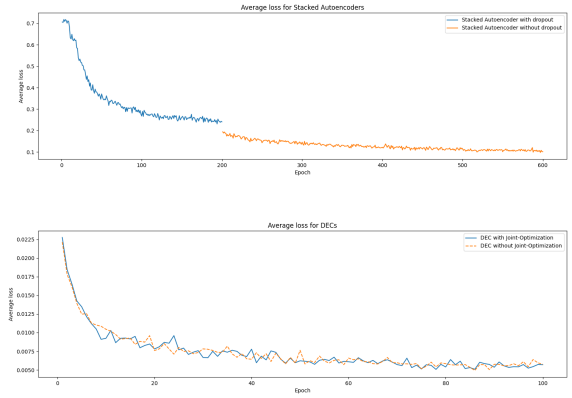


Abbildung B.1: Verlustfunktion des DECs auf dem Datensatz MNIST

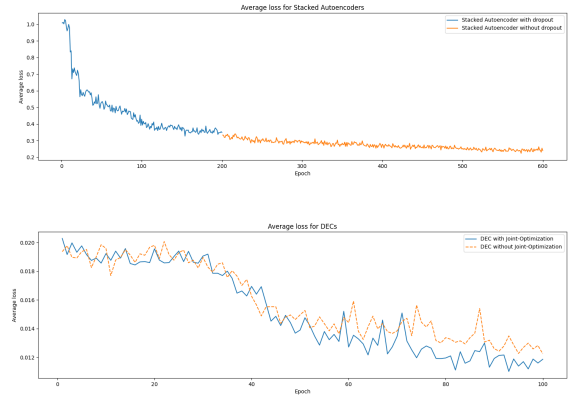
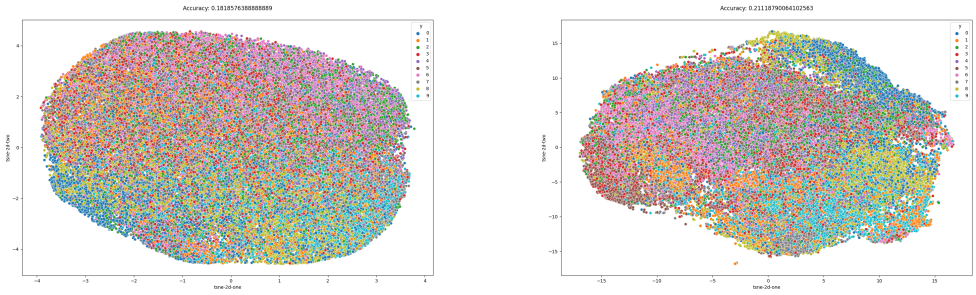


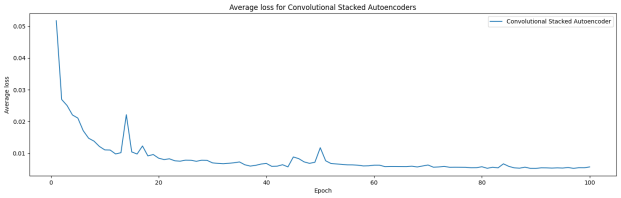
Abbildung B.2: Verlustfunktion des DECs auf dem Datensatz CIFAR10



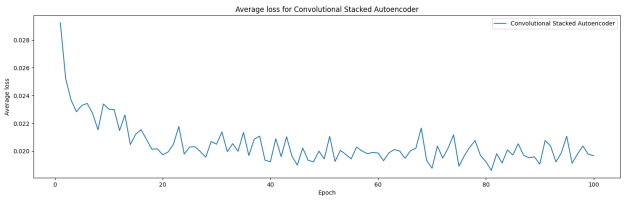
(a) SCA mit Pooling-Schichten

(b) SCA mit Strides

Abbildung B.3: (Kaptel 6.3.4) Illustrierung anhand des TSNE Verfahrens für zwei Stacked Convolutional Autoencoder (SCA) auf dem **Datensatz CIFAR10**



(a) Verlauf der durchschnittlichen Verluste von SCA mit Pooling-Schichten



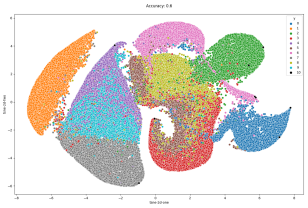
(b) Verlauf der durchschnittlichen Verluste von SCA mit Strides

Abbildung B.4: (Kaptel 6.3.4) Darstellung des durschnittlichen Verlustes der jeweiligen Epoche für zwei Stacked Convolutional Autoencoder (SCA) auf dem **Datensatz CIFAR10**

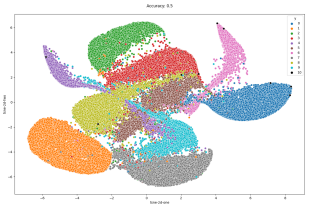
Anhang C

Anhang III

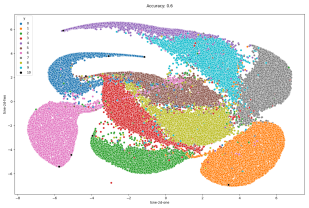
C.1 Abbildungen zum Experiment II



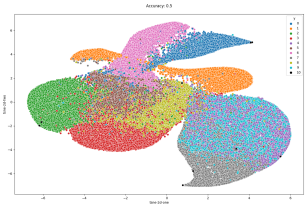
(a) Nach Epoche 50
$\alpha = 3.1$ $\lambda = 0.05$



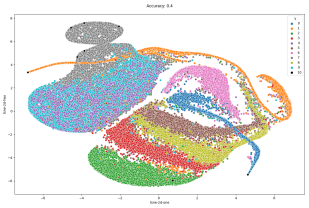
(b) Nach Epoche 300
$\alpha = 3.1$ $\lambda = 0.05$



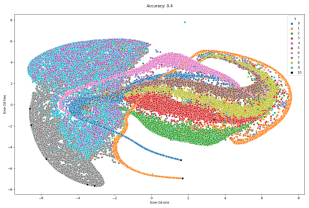
(c) Nach Epoche 700
$\alpha = 3.1$ $\lambda = 0.05$



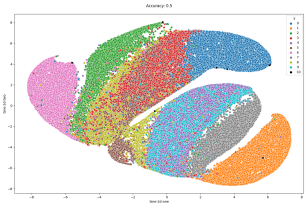
(d) Nach Epoche 50
$\alpha = 3.1$ $\lambda = 0.1$



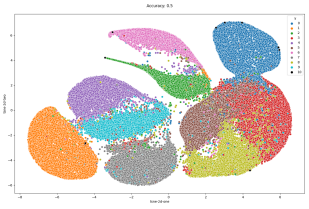
(e) Nach Epoche 300
$\alpha = 3.1$ $\lambda = 0.1$



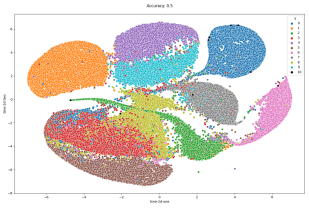
(f) Nach Epoche 700
$\alpha = 3.1$ $\lambda = 0.1$



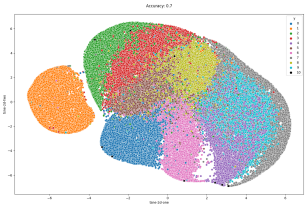
(g) Nach Epoche 50
$\alpha = 5.5$ $\lambda = 0.05$



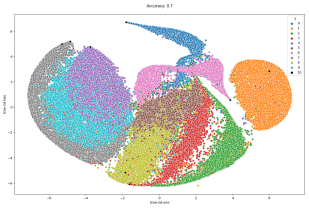
(h) Nach Epoche 300
$\alpha = 5.5$ $\lambda = 0.05$



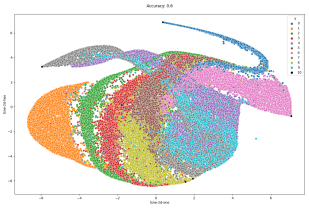
(i) Nach Epoche 700
$\alpha = 5.5$ $\lambda = 0.05$



(j) Nach Epoche 50
$\alpha = 5.5$ $\lambda = 0.1$

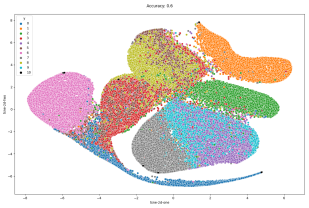


(k) Nach Epoche 300
$\alpha = 5.5$ $\lambda = 0.1$

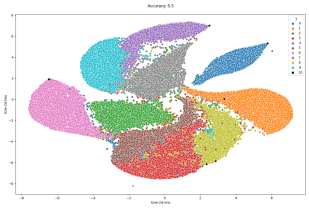


(l) Nach Epoche 700
$\alpha = 5.5$ $\lambda = 0.1$

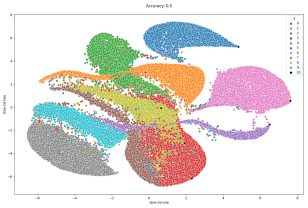
Abbildung C.1: (Kapitel 6.4.2) TSNE-Abbildungen zur Darstellung des Verlaufs von DSA für den Kernel-Parameter $\alpha = 3.1, 5.5$ und den Regulierungsfaktor $\lambda = 0.05, 0.1$ und ACC-Accuracy bezogen auf die aktuelle Zusammenfassung



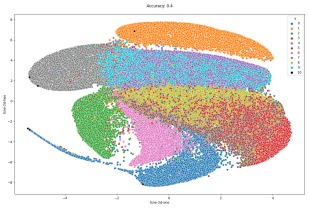
(a) Nach Epoche 50
$\alpha = 8.0$ $\lambda = 0.05$



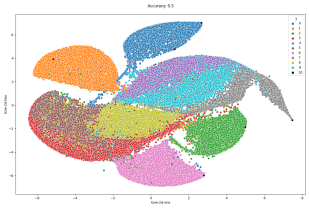
(b) Nach Epoche 300
$\alpha = 8.0$ $\lambda = 0.05$



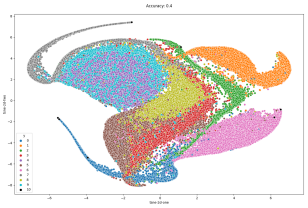
(c) Nach Epoche 700
$\alpha = 8.0$ $\lambda = 0.05$



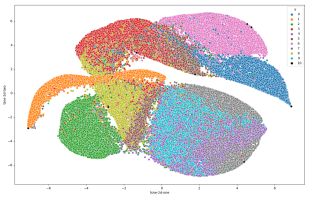
(d) Nach Epoche 50
$\alpha = 8.0$ $\lambda = 0.1$



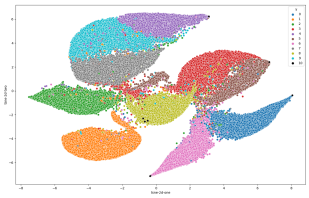
(e) Nach Epoche 300
$\alpha = 8.0$ $\lambda = 0.1$



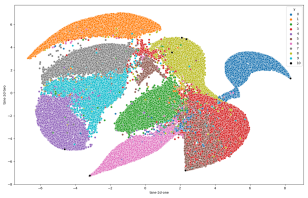
(f) Nach Epoche 700
$\alpha = 8.0$ $\lambda = 0.1$



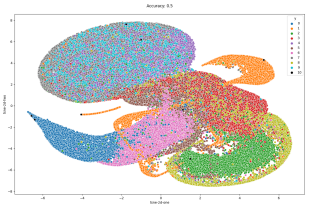
(g) Nach Epoche 50
$\alpha = 10.0$ $\lambda = 0.05$



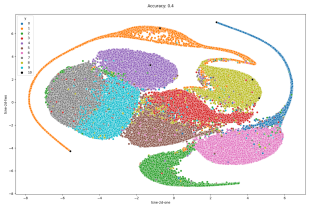
(h) Nach Epoche 300
$\alpha = 10.0$ $\lambda = 0.05$



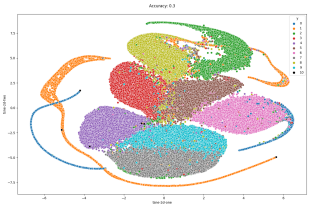
(i) Nach Epoche 700
$\alpha = 10.0$ $\lambda = 0.05$



(j) Nach Epoche 50
$\alpha = 10.0$ $\lambda = 0.1$

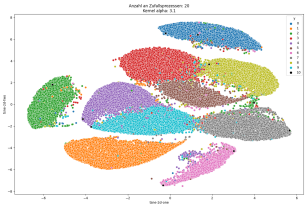


(k) Nach Epoche 300
$\alpha = 10.0$ $\lambda = 0.1$

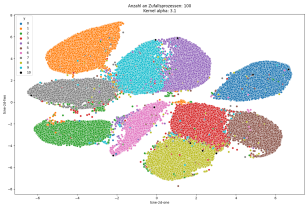


(l) Nach Epoche 700
$\alpha = 10.0$ $\lambda = 0.1$

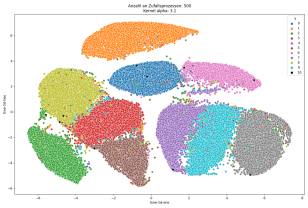
Abbildung C.2: (Kapitel 6.4.2) TSNE-Abbildungen zur Darstellung des Verlaufs von DSA für den Kernel-Parameter $\alpha = 8.0, 10.0$ und den Regulierungsfaktor $\lambda = 0.05, 0.1$ und ACC-Accuracy bezogen auf die aktuelle Zusammenfassung



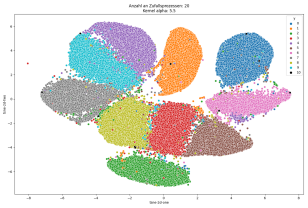
(a) 20 Zufallsprozesse, $\alpha = 3.1$



(b) 100 Zufallsprozesse, $\alpha = 3.1$

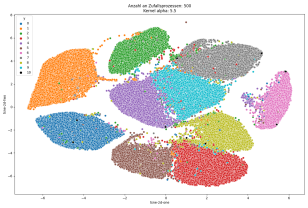


(c) 500 Zufallsprozesse, $\alpha = 3.1$

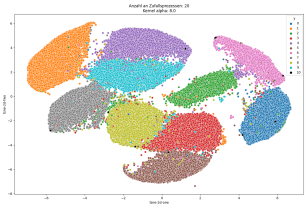


(d) 20 Zufallsprozesse, $\alpha = 5.5$

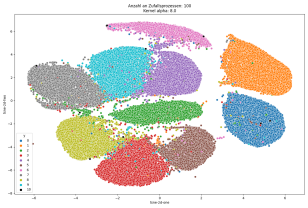
(e) 100 Zufallsprozesse, $\alpha = 5.5$



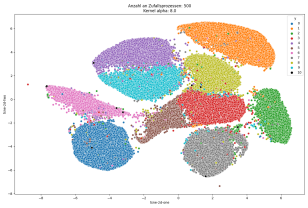
(f) 500 Zufallsprozesse, $\alpha = 5.5$



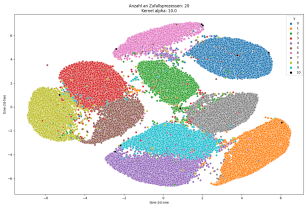
(g) 20 Zufallsprozesse, $\alpha = 8.0$



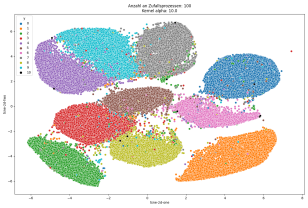
(h) 100 Zufallsprozesse, $\alpha = 8.0$



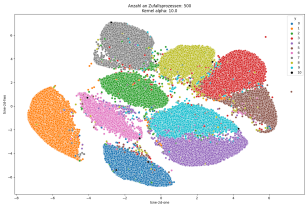
(i) 500 Zufallsprozesse, $\alpha = 8.0$



(j) 20 Zufallsprozesse, $\alpha = 10.0$



(k) 100 Zufallsprozesse, $\alpha = 10.0$



(l) 500 Zufallsprozesse, $\alpha = 10.0$

Abbildung C.3: (Kapitel 6.4.3) Illustrierung der eingebetteten Räume anhand des TSNE-Verfahrens für verschiedene Zufallsprozesse. Für die selben α Wert, handelt es sich um die selben Räume.

Abbildungsverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

[1] Aggarwal, Charu C., Alexander Hinneburg und Daniel A. Keim: *On the Surprising Behavior of Distance Metrics in High Dimensional Space.* In: Bussche, Jan Van den und Victor Vianu (Herausgeber): *Database Theory — ICDT 2001*, Seiten 420–434, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[2] Badanidiyuru, Ashwinkumar, Baharan Mirzasoleiman, Amin Karbasi und Andreas Krause: *Streaming Submodular Maximization: Massive Data Summarization on the Fly.* 01 2014.

[3] Bekkersr, Erik: *Lecture 12.5: Gaussian Processes: Regression, Course: Machine Learning 1.* University Lecture, 2020.

[4] Bellman, Richard E.: *Adaptive control processes - A guided tour.* Princeton University Press, Princeton, New Jersey, U.S.A., 1961.

[5] Belyaev, Mikhail, Evgeny Burnaev und Yermek Kapushev: *Exact Inference for Gaussian Process Regression in case of Big Data with the Cartesian Product Structure.* 03 2014.

[6] Bengio, Yoshua, Aaron Courville und Pascal Vincent: *Representation Learning: A Review and New Perspectives.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(8):1798–1828, 2013.

[7] Beyer, Kevin, Jonathan Goldstein, Raghu Ramakrishnan und Uri Shaft: *When Is "Nearest Neighbor" Meaningful?* In: Beeri, Catriel und Peter Buneman (Herausgeber): *Database Theory — ICDT'99*, Seiten 217–235, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[8] Bilmes, J. und Hui-Ching Lin: *Submodularity in natural language processing: algorithms and applications.* 2012.

[9] Bilmes, Jeffrey A. und Rishabh Iyer: *NOML: Submodularity in Machine Learning.* University Lecture, June 2015.

[10] Buschjäger, Sebastian: *Online Gauß-Prozesse zur Regression auf FPGAs*. Master's Thesis, TU Dortmund University, 2016.

[11] Buschjäger, Sebastian, Philipp-Jan Honysz, Lukas Pfahler und Katharina Morik: *Very Fast Streaming Submodular Function Maximization*, 2020.

[12] Carbonell, Jaime und Jade Goldstein: *The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries*. In: *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, Seite 335–336, New York, NY, USA, 1998. Association for Computing Machinery.

[13] CAUCHY, A.: *Methode generale pour la resolution des systemes d'equations simultanees*. C.R. Acad. Sci. Paris, 25:536–538, 1847.

[14] Chapelle, Olivier, Bernhard Schölkopf und Alexander Zien: *Semi-Supervised Learning*. 09 2006.

[15] Dahal, Paras: *Deep Clustering*. https://deepnotes.io/deep-clustering.

[16] developers scikit-learn: *2.1. Gaussian mixture models*, 2007 - 2021. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html.

[17] Durán, Jaime: *Everything You Need to Know about Gradient Descent Applied to Neural Networks*, September 2019. https://medium.com/yottabytes.

[18] Eibe Frank, Ian Witten nad, Mark Hall und Christopher Pal: *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, October 2016.

[19] Elhamifar, Ehsan, Amit Roy-Chowdhury und Amin Karbasi: *Big Data Summarization: Algorithms and Applications*. Tutorial at 2018 Conference on Computer Vision and Pattern Recognition, 2018.

[20] Erdem, Kemal: *t-SNE clearly explained An intuitive explanation of t-SNE algorithm and why it's so useful in practice*. Towards Data Science, April 2020.

[21] Feige, Uriel: *A Threshold of Ln n for Approximating Set Cover*. J. ACM, 45(4):634–652, Juli 1998.

[22] Frieze, A. M.: *A cost function property for plant location problems*. Mathematical Programming, 7:245–248, 1974.

[23] Gomes, Ryan und Andreas Krause: *Budgeted Nonparametric Learning from Data Streams*. Seiten 391–398, 07 2010.

[24] Guo, Xifeng, Xinwang Liu, En Zhu und Jianping Yin: *Deep Clustering with Convolutional Autoencoders*. Seiten 373–382, 10 2017.

[25] Guo, Xifeng, Xinwang Liu, En Zhu und Jianping Yin: *Deep Clustering with Convolutional Autoencoders*. Seiten 373–382, 10 2017.

[26] Hesabi, Z. R., Z. Tari, A. Goscinski, A. Fahad, I. Khalil und C. Queiroz: *Data Summarization Techniques for Big Data—A Survey*, Seiten 1109–1152. Springer New York, New York, NY, 2015.

[27] Hinton, G. E. und R. R. Salakhutdinov: *Reducing the Dimensionality of Data with Neural Networks*. Science, 313(5786):504–507, 2006.

[28] Hinton, Geoffrey E und Sam Roweis: *Stochastic Neighbor Embedding*. In: Becker, S., S. Thrun und K. Obermayer (Herausgeber): *Advances in Neural Information Processing Systems*, Band 15. MIT Press, 2003.

[29] Hornik, Kurt: *Approximation capabilities of multilayer feedforward networks*. Neural Networks, 4(2):251–257, 1991.

[30] Huang, Xiaowei, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu und Xinping Yi: *A Survey of Safety and Trustworthiness of DeepNeural Networks: Verification, Testing, Adversarial Attack and Defence, and Interpretability*, Februar 2018. arXiv, page arXiv:1812.08342v5.

[31] Kaufman, Leonard und Peter J. Rousseeuw: *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley, 1990.

[32] Krause, Andreas und Daniel Golovin: *Submodular function maximization*. Tractability: Practical Approaches to Hard Problems, 3(19):8, 2012.

[33] Krizhevsky, Alex: *Learning Multiple Layers of Features from Tiny Images*. University of Toronto, 05 2012.

[34] Kuhn, H. W.: *The Hungarian method for the assignment problem*. Naval Research Logistics Quarterly, 2(1-2):83–97, 1955.

[35] Lawrence, Neil, John Platt und Michael Jordan: *Extensions of the Informative Vector Machine*. Band 3635, Seiten 56–87, 01 2004.

[36] Lawrence, Neil D, Matthias Seeger und Ralf Herbrich: *Fast sparse Gaussian process methods: The informative vector machine*. In: *Advances in neural information processing systems*, Seiten 625–632, 2003.

[37] Le, Quoc V., Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeff Dean und Andrew Y. Ng: *Building high-level features using large scale unsupervised learning*, 2012.

[38] Lecun, Yann: *PhD thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models)*. Universite P. et M. Curie (Paris 6), Juni 1987.

[39] Lecun, Yann, Leon Bottou, Y. Bengio und Patrick Haffner: *Gradient-Based Learning Applied to Document Recognition*. Proceedings of the IEEE, 86:2278 – 2324, 12 1998.

[40] Li, Fengfu, Hong Qiao, Bo Zhang und Xuanyang Xi: *Discriminatively Boosted Image Clustering with Fully Convolutional Auto-Encoders*, 2017.

[41] Lin, Hui und Jeff Bilmes: *A class of submodular functions for document summarization*. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, Seiten 510–520. Association for Computational Linguistics, 2011.

[42] Liu, Guifang, Huaiqian Bao und Baokun Han: *A Stacked Autoencoder-Based Deep Neural Network for Achieving Gearbox Fault Diagnosis*. Mathematical Problems in Engineering, 2018, Jul 2018.

[43] Luber, Stefan: *Was ist ein Neuronales Netz?* https://www.bigdata-insider.de/was-ist-ein-neuronales-netz-a-686185/, Feb 2018.

[44] Maaten, Laurens van der und Geoffrey Hinton: *Visualizing data using t-SNE*. Journal of Machine Learning Research, 9:2579–2605, 11 2008.

[45] MacQueen, J.: *Some Methods for Classification and Analysis of Multivariate Observations*. In: Le Cam, L. M. und J. Neyman (Herausgeber): *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability - Vol. 1*, Seiten 281–297. University of California Press, Berkeley, CA, USA, 1967.

[46] Masci, Jonathan, Ueli Meier, Dan C. Ciresan und Jürgen Schmidhuber: *Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction*. In: *ICANN*, 2011.

[47] Minoux, Michel: *Accelerated greedy algorithms for maximizing submodular set functions*. In: Stoer, J. (Herausgeber): *Optimization Techniques*, Seiten 234–243, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg.

[48] Mirzasoleiman, Baharan, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák und Andreas Krause: *Lazier than Lazy Greedy*. AAAI'15, Seite 1812–1818. AAAI Press, 2015.

[49] Morik, Katharina: *Maschinelles Lernen: Skript zur Vorlesung.* University Lecture: Technische Universität Dortmund, September 2013.

[50] Nemhauser, G. L., L. A. Wolsey und M. L. Fisher: *An analysis of approximations for maximizing submodular set functions—I.* Mathematical Programming, 14(1):265–294, Dec 1978.

[51] Rasmussen, Carl Edward und Christopher K. I. Williams: *Gaussian processes for machine learning.* Adaptive computation and machine learning. MIT Press, 2006.

[52] Ruder, Sebastian: *An overview of gradient descent optimization algorithms*, 2017.

[53] Rüping, Stefan: *SVM Kernels for Time Series Analysis.* In: *LLWA 01 - Tagungsband der GI-Workshop-Woche Lernen - Lehren - Wissen - Adaptivitat*, Forschungsberichte des Fachbereichs Informatik der Universitat Dortmund, Seiten 43–50, October 2001.

[54] Sakurada, Mayu und Takehisa Yairi: *Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction.* MLSDA'14, Seite 4–11, New York, NY, USA, 2014. Association for Computing Machinery.

[55] Schmidt, Maik: *Datenzusammenfassungen auf Datenströmen.* Bachelor's Thesis, TU Dortmund University, 2017.

[56] Seeger, Matthias: *Greedy Forward Selection in the Informative Vector Machine.* 01 2004.

[57] Vincent, Pascal, Hugo Larochelle, Yoshua Bengio und Pierre-Antoine Manzagol: *Extracting and Composing Robust Features with Denoising Autoencoders.* In: *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, Seite 1096–1103, New York, NY, USA, 2008. Association for Computing Machinery.

[58] Vincent, Pascal, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio und Pierre-Antoine Manzagol: *Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion.* Journal of Machine Learning Research, 11(110):3371–3408, 2010.

[59] Wood, Thomas: *What is the Sigmoid Function?* https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function.

[60] Xie, Junyuan, Ross Girshick und Ali Farhadi: *Unsupervised Deep Embedding for Clustering Analysis.* 33rd International Conference on Machine Learning, ICML 2016, 1:740–749, 2016.

[61] Zhai, Junhai, Sufang Zhang, Junfen Chen und Qiang He: *Autoencoder and Its Various Variants.* In: *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Seiten 415–419, 2018.

[62] Zhou, Chong und Randy C. Paffenroth: *Anomaly Detection with Robust Deep Autoencoders*. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Seite 665–674, New York, NY, USA, 2017. Association for Computing Machinery.