Hochschule Bochum
Bochum University
of Applied Sciences

Hochschule Bochum
Fachbereich Wirtschaft

Bachelorarbeit

Variational Autoencoder zur Optimierung von Videos

Erstprüfer: Prof. Dr. …

Zweitprüfer: Prof. Dr. …

Jana Mustermann

Matrikelnummer: 123456789

Musterstraße. 1

44801 Bochum

Tel.: 0234/123456

E-Mail: jana.mustermann@hs-bochum.de

Studiengang:

Wirtschaftsinformatik

9. Fachsemester, Wintersemester 2018

Abgabedatum 21. Dezember 2018

Inhaltsverzeichnis

Abkürzungsverzeichnis

ELBO ..... Evidence Lower Bound

EM-Algorithmus. ..... Expectation-Maximization-Algorithmus

GAN ..... Generative Adversarial Network

KL-Abweichung ..... Kullback-Leibler-Abweichung

MLE ..... Maximum Likelihood Estimation

MSE ..... Mean Squared Error

ReLU. ..... Rectified Linear Unit

VAE. ..... Variational Autoencoder

VI ..... Variational Inference

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Welt wird immer komplexer! Die Welt hat sich in dem vergangenen Jahrhundert von einem Industriezeitalter zu einem Informationszeitalter entwickelt. Bereits heute ist es möglich, durch das Internet an eine schier unüberschaubare Fülle an Informationen zu kommen. Diese Menge ist so groß, dass ein einzelner Mensch diese unmöglich allein verarbeiten könnte. Die Menge allein sagt jedoch relativ wenig über den Informationsgehalt der einzelnen Nachrichten aus. Beispielsweise bietet gute Fachlektüre einen umfassenden Überblick, geht im Detail auf die einzelnen Subthemen ein und verweist auf andere Literatur, so dass es möglich, ist sich weiter mit dem Thema auseinanderzusetzen. Der bzw. die Autoren haben die wesentlichen Informationen aus verschiedenen Quellen gebündelt und so aufbereitet, dass die Informationen im entsprechenden Kontext einen logischen Gedankengang ausbauen. Prinzipiell werden Informationen zum Erschließen von neuen Informationen benötigt. Das Erschließen von neuen Informationen lässt sich beispielsweise an zahlreichen Dienstleistungen beobachten. Börsenprofis, die zweifelhafte Tipps für das Investieren geben oder auch Juristen, die die wesentlichen Aspekte eines Falls auswerten, verdichten eine Menge an Informationen und Indikatoren und leiten daraus eine neue Information ab, welches ebenfalls eine neue Information darstellt, die weiter verwendet werden kann. Wir können Kriterien für diese neuen Informationen aufstellen. Wir wollen beispielsweise, dass die neue Information eine verdichtete Information ist, da wir anstatt vieler Informationen nur eine einzige auszuwerten brauchen, da die schiere Masse an Daten uns nur erlaubt, einen sehr geringen Bruchteil davon auszuwerten. Wir wollen außerdem, dass die Daten relevant und hilfreich sind, um auf deren Basis kluge Entscheidungen zu treffen. Sollen wir die Aktie verkaufen oder behalten? Diese zwei Kriterien benötigen einerseits Daten und anderseits eine Möglichkeit, diese Informationen zu verdichten. Eine Technologie, die das verspricht, ist das maschinelle Lernen. Diese Technologie erlebt seit den letzten Jahrzehnten einen regelrechten Boom und Experten stellen das maschinelle Lernen als eine Art Wunderwaffe vor und nahezu täglich strömen neue Impulse an Ideen in die Entwicklung neuer verbesserter Verfahren und Algorithmen.

Wenn die Computer ein besseres Verständnis über die Daten besitzen und intelligenter agieren können als ein bloßes Regelwerk von Anweisungen, wie bei einer imperativen Programmierung der Fall, würden sich neue Möglichkeiten auftun. Neue Entdeckungen könnten gemacht werden, neue Produkte könnten erstellt werden und es könnte eine komplett neue Welt entstehen. Die

Auswirkungen dieser Technologie regen seit Generationen zahlreiche Science-Fiction-Autoren zum Fantasieren an, die von einer Welt von morgen träumen.
Wir wollen uns in dieser Arbeit einen generellen Überblick über den aktuellen Wissensstand des maschinellen Lernens verschaffen und das maschinelle Lernen in Verbindung mit der Statistik einsetzen, um die Bildwiederholungsrate von Videos nachträglich zu erhöhen. Diese Fragestellung ist insofern interessant, da dies nur möglich ist, wenn der Algorithmus ein Verständnis über die Daten bzw. die gezeigten Bilder gewinnt, da die einzelnen Bildpunkte richtig verschoben werden müssen, um ein in sich stimmiges Video zu erhalten. Wir werden uns daher mit der Frage beschäftigen, inwieweit die Technologie des Variational Autoencoders für diese Problemstellung eingesetzt werden kann.

Die Arbeit ist dabei in vier Hauptkapitel eingeteilt. Die ersten drei Kapitel befassen sich mit den theoretischen Grundlagen, die für das vierte Kapitel, dem Praxisprojekt, nötig sind. Wir werden uns zunächst mit den Grundlagen des maschinellen Lernens befassen und anschließend auf die statistischen Grundlagen eingehen, die notwendig sind um Schlüsse aus den Daten ziehen zu können. Die Kombination dieser beider Themen erlaubt es, anspruchsvollere Konzepte des maschinellen Lernens umzusetzen, wie beispielsweise den Variational Autoencoder, und bildet das letzte Kapitel des theoretischen Teils dieser Arbeit.

2 Maschinelles Lernen

Hinter dem Begriff Maschinelles Lernen (engl. Machine Learning) verstecken sich Algorithmen die fähig sind von Daten zu lernen, um mit neuen gleichartigen Daten umzugehen.
Mitchell liefert eine Definition zu dem Lernprozess:
„A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."[1]
Dabei ist die Erfahrung E eine Sammlung von Datensätzen, die von der Aufgabe abhängig sind, wie beispielsweise Sensordaten, Bilder, Texte oder Ähnliches. Die Aufgabe T beschreibt was der Algorithmus tun soll wie beispielsweise Werte vorhersagen, Datensätze klassifizieren, Bilder generieren usw. Wie gut der Algorithmus diese Aufgaben ausführt wird durch die Messung von P bestimmt. P wird häufig auch als Fehlerfunktion, Verlustfunktion oder auch als Zielsetzungsfunktion bezeichnet.[2]
Generell wird zwischen drei verschiedenen übergeordneten Lerntypen unterschieden:

- Überwachtes Lernen: Dabei hat jeder Datensatz neben den eigentlichen Daten noch den gewünschten Output bzw. ein Label. Das Ziel ist es eine Funktion zu finden die den Datensatz auf den Output mappt. Der Term „überwacht" entwickelte sich daraus, dass ein „Lehrer" oder auch Supervisor dem Algorithmus beim Lernen hilft und ihm den richtigen Wert vorgibt. Das überwachte Lernen stellt das Herz des maschinellen Lernens da und viele Algorithmen des maschinellen Lernens versuchen, wie wir später sehen werden, die Aufgabe auf eine Problemstellung des überwachten Lernens zurückzuführen.
- Unüberwachtes Lernen: Beim unüberwachten Lernen wird dem Algorithmus kein entsprechendes Zielattribut vorgegeben. Es geht überwiegend darum verborgenes Wissen bzw. Informationen zu extrahieren und dem Anwender tiefere Einblicke zu gewähren, um Schlüsse über Verhalten oder Kausalitäten aufzudecken. Es ist eng verwandt mit der Mustererkennung und Data Mining. Eine Besonderheit an dieser Problemstellung ist, dass die Anzahl der verfügbaren Daten enorm ist im Vergleich zu dem überwachten Lernen, in welchem der Output vorgegeben wird. Unüberwachtes Lernen wird auch als „Information Retrieval"-Prozess bezeichnet.
- Bestärkendes Lernen: Eine weitere Lernform ist das bestärkende Lernen. Dabei interagiert der Algorithmus mit der Umgebung und erhält Inputreize aufgrund von vorherigen

[1] Mitchell (1997), S. 2.
[2] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 82.

Handlungen in einer Umgebung. Ziel ist es, basierend auf dem Muster von Inputreizen entsprechende Aktionen bzw. Reaktionen auszuführen mit dem Ziel, seine Belohnungen (engl. rewards) zu maximieren. Der Algorithmus lernt mit fortschreitendender Simulation, wie gut oder wie schlecht eine Aktion war und passt seine Aktionen dementsprechend an.[3]

Das Lernen im eigentlichen Sinne geschieht meistens dadurch, dass der Algorithmus durch eine Reihe von Parametern θ sein Verhalten gegenüber den Inputmustern bestimmt wird. Durch gezielte Beeinflussung dieser Parameter erlernt der Algorithmus das gewünschte Verhalten. Zusammengefasst lässt sich also sagen, dass wir Erfahrungen bzw. Daten brauchen sowie ein Modell bzw. einen Algorithmus, der diese Daten verarbeitet, und eine Performance-Funktion im Bezug auf die Aufgabe.

2.1 **Modelle des überwachten Lernens**

Ziel des überwachten Lernens ist es, ein Model zu finden das in der Lage ist, auf neue, unbekannte Daten richtig zu reagieren.[4] Damit ein Modell diese Anforderung erfüllt, führen wir die Begriffe Unteranpassung, Überanpassung und die Fähigkeit zu generalisieren ein, um Modelle nach diesen Punkten zu bewerten. Als Beispiel nehmen wir uns ein Regressionsproblem, bei dem wir versuchen, eine Reihe von Datenpunkten mit einer Funktion zu approximieren, sodass die Punkte auf oder möglichst nahe an dem Graphen der Funktion liegen. Das Modell erhält eine Eingabe x und generiert dazu den geschätzten Wert für das Zielattribut y.

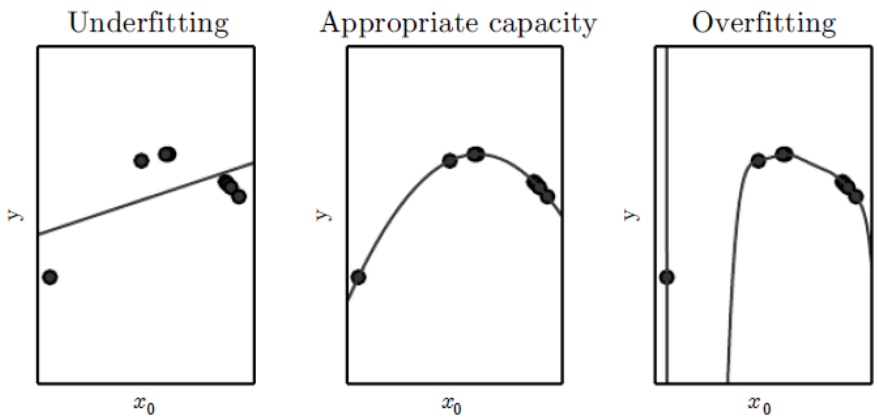


Abbildung 1: Unter- und Überanpassung anhand eines Regressionsproblems
Quelle: Goodfellow/ Bengio/ Courville (2016), S. 113.

[3] Vgl. Kriesel (2007), S. 53.
[4] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 110.

Es ist sehr deutlich erkennbar, dass der Graph auf der linken Seite die ursprünglichen Daten suboptimal abbildet. Das liegt daran, dass das Modell versucht, eine nicht lineare Funktion mit einer linearen Funktion zu approximieren. Allerdings liegt der Fehler nicht an dem Modell, sondern an dem Anwender, der den Daten einen linearen Verlauf unterstellt und das Modell falsch designt hat. Der Graph auf der rechten Seite zeigt eine Überanpassung des Graphen. Dabei wird versucht, ein Polynom höheren Grades auf ein Polynom kleineren Grades abzubilden. Wenn wir die Funktion nutzen würden um einen neuen Wert abzubilden, würden wir untypische Werte erhalten, die zwar den Trainingsdaten entsprechen, allerdings nicht den neuen Daten. Die Abbildung in der Mitte zeigt ein Modell, dass eine Funktion gelernt hat, welche die Daten am ehesten repräsentiert. Eine Aufgabe ist es, einen guten Trade-off zwischen einer Unter- und Überanpassung zu finden, um die Funktion möglichst exakt zu approximieren. Generell lässt sich sagen, dass die Komplexität der Aufgabe ungefähr der Komplexität des Modells entsprechen soll.[5] Um die Fähigkeit des Generalisierens zu testen werden in der Praxis häufig gezielt Datensätze entfernt bzw. dem Model vorenthalten, um das Modell auf genau diese Eigenschaften hin zu überprüfen. Diese Überprüfung kann beispielsweise durch die Performancemessung stattfinden, sofern die Labels bekannt sind, um einen Fehlerwert zu berechnen und diese miteinander zu vergleichen.

2.2 **Algorithmen des überwachten Lernens**

Es existiert eine Vielzahl an Modellen, um die Aufgabenstellung des überwachten Lernens zu lösen. Je nach Menge der Daten, der Problemstellung sowie der Verfügbarkeit von Rechenressourcen hängt die Wahl des entsprechenden Algorithmus davon ab. Wir werden in dieser Arbeit zwei gängige Vertreter betrachten, die für eine Vielzahl an Problemstellungen einsetzbar sind, nämlich künstliche neuronale Netze, die sich insbesondere durch ihre vielfältigen Einsatzmöglichkeiten auszeichnen, und konvolutionale Netze, die meist in der Bilderkennung eingesetzt werden.

2.2.1 Neuronale Netze

In dem vergangenen Jahrzehnt haben sich künstliche neuronale Netze als effiziente Möglichkeit herausgestellt, um nicht lineare Funktionen zu approximieren.[6] Wie der Name bereits vermuten lässt, war die ursprüngliche Idee dahinter, das zentrale Nervensystem auf einer mathematischen Ebene abzubilden. Zu den Pionieren auf diesem Themenfeld gehören Warren McCulloch und

[5] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 112.
[6] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 224.

Walter Pitts, die in ihrer Arbeit im Jahr 1943 erste Ideen für ein vorwärts gerichtetes neuronales Modell vorstellen.[7]

Die Idee hinter neuronalen Netzen ist, dass ein Neuron durch eine Reihe von Inputs aktiviert wird bzw. Informationen erhält, diese dann durch eine mathematische Vorschrift miteinander kombiniert und anschließend den berechneten Wert weitergibt. Wir werden uns in diesem Kapitel mit den wesentlichen Elementen von neuronalen Netzen vertraut machen, die in einer Problemstellung des überwachten Lernens angewendet werden.

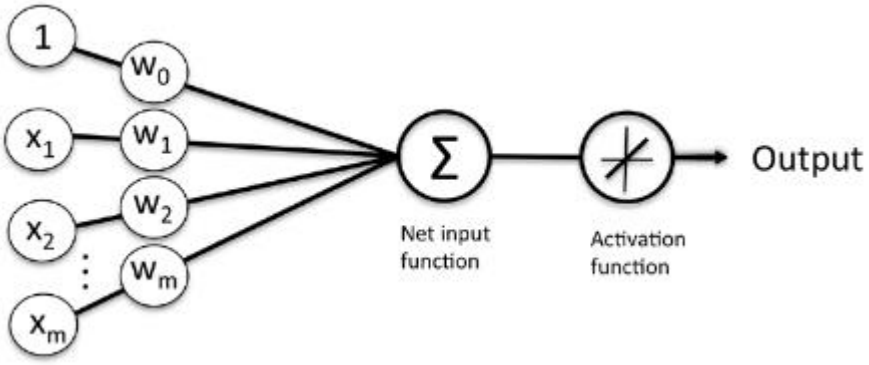


Abbildung 2: Neuronales Netz mit einem einzigen Neuron
Quelle: in Anlehnung an Raschka (2017), S. 42.

Die Abbildung zeigt den schematischen Aufbau eines neuronalen Netzes mit einer Inputschicht und einem Ausgabeneuron bzw. einer Ausgabeschicht mit einem einzigen Neuron. Auf der linken Seite befinden sich die Inputreize in Form eines Vektors $x \in \mathbb{R}^n$, die mit einem Gewichtsvektor $w \in \mathbb{R}^n$ multipliziert werden und so den vorläufigen Output des Neurons darstellen. Es wird noch ein Offset-Bias-Parameter W_0 hinzuaddiert (im Nachfolgenden mit b abgekürzt), da der vorhergesagte Graph sonst nur durch den Ursprung laufen würde.[8] Beispielsweise könnte die Funktion $f(x) = 1$ nicht abgebildet werden ohne den Bias-Parameter. Abschließend wird noch eine sogenannte Aktivierungsfunktion angewandt, um nicht-lineare Funktionen abbilden zu können. Die Gewichte lassen sich wie folgt interpretieren: Ein negatives Gewicht ist ein für das Neuron hemmender Reiz, das die Aktivierung des Neurons mindern soll. Ein positives Gewicht hingegen hat einen erregenden Einfluss und soll das Neuron in einen aktiven Status versetzen. Wenn das Gewicht nahe Null ist übt es keine oder nur eine geringe Wirkung aus, es hat also weder eine animierende noch eine hemmende Wirkung.[9]

Diesem Modell ist es allerdings nur beschränkt möglich, eine beliebig komplexe Funktion zu erlernen, deswegen wird in dem Kontext von neuronalen Netzen häufig der Begriff „Deep

[7] McCulloch/ Pitts (1943), S. 113-133.
[8] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 109.
[9] Vgl. Rey/Wender (2011), S. 15.

Learning“ verwendet. Es handelt sich dabei um die Aneinanderreihung von neuronalen Netzschichten, um komplexe geschachtelte Funktionen zu erlernen. [10]

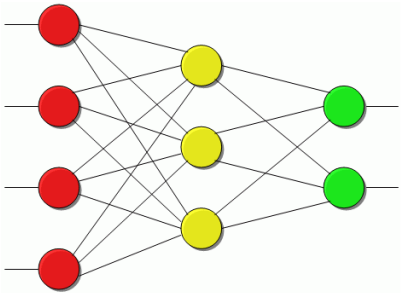


Abbildung 3: Der Aufbau eines tiefen neuronalen Netzes
Quelle: Rey (2011), S. 15.

Die Abbildung zeigt ein tiefes neuronales Netz mit einer Inputschicht (rot), einer versteckten Schicht (gelb) und einer Ausgabeschicht (grün). Die mathematische Vorschrift zu Berechnung der Ausgabe sieht wie folgt aus:

$$f(x) = f_2\big(f_1(x)\big)\, mit\, f_i(x) = \ activation(W_i\, x + \ b_i)$$

Jede Schicht bekommt eine Gewichtsmatrix W und einen Bias-Parameter b zugeordnet, mit denen sich die Eingaben einer höheren Schicht berechnen lassen. Diese Prozedur wird auch als ein vorwärts gerichtetes Propagieren bezeichnet.[11]

Das Designen von versteckten Schichten wie beispielsweise die Auswahl von Aktivitätsfunktionen sowie die Festlegung der Anzahl der Neuronen pro Schicht wird im Zusammenhang mit der Struktur, mit der die einzelnen Schichten in Verbindung stehen, auch als Modellarchitektur bezeichnet.[12] Die Auswahl wird vor dem Trainingsprozess festgelegt und kann in der Regel nicht nachträglich beeinflusst werden.

Ein weiterer wichtiger Faktor ist die Wahl der Aktivierungsfunktionen. Üblicherweise werden in dem gesamten Netzwerk die gleichen Aktivierungsfunktionen verwendet, außer der Ausgabeschicht, in der diese nach den Bedürfnissen angepasst werden (wie beispielsweise der Wertebereich der Ausgabe). Dabei ist es in den meisten Fällen so, dass zwingend Aktivierungsfunktionen benötigt werden, um die Mächtigkeit des Netzwerkes zu erhöhen. Präziser ausgedrückt werden Aktivierungsfunktionen benötigt, um nicht-lineare Funktionen approximieren zu können. Um deren Relevanz zu zeigen modifizieren wir die Funktion für die vorwärts gerichtete Propagation. In ihr wird auf die Aktivierungsfunktion verzichtet sowie auf den Bias-Parameter, um die Gleichung übersichtlicher zu gestalten.

$$f(x) = f_2\big(f_1(x)\big)\, mit\, f_i(x) = \ W_i x$$

[10] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 1 f.
[11] Vgl. Rey/Wender (2011), S. 52.
[12] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 197.

Anders formuliert entsteht:

$$f(x) = W_2 W_1 x = W^* x$$

Das tiefe neuronale Netz hat also nicht die Fähigkeit, komplexere Funktionen zu erlernen, da sie mit einer einzigen Gewichtsmatrix W* ausgetauscht werden kann. Es werden demzufolge zwingend Aktivierungsfunktionen benötigt, um nicht-lineare Funktionen zu erlernen.[13]

Prinzipiell erfüllt jede Funktion diese Eigenschaft, die nicht ausschließlich linear ist. Allerdings haben sich in der Literatur sowie in der Praxis einige Aktivitätsfunktionen etabliert, wobei in diesem Bereich noch aktiv Forschung betrieben wird. Die Auswahl kann nach verschiedenen Kriterien erfolgen, wie beispielsweise der Begrenzung des Wertebereichs, der biologischen Plausibilität oder aber auch die Differenzierbarkeit der Funktion.[14] Eine weitere interessante Eigenschaft ist, wie sich die Ableitung der Funktion verhält, also ob der Gradient auf bestimmten Teilbereichen der Funktion sehr klein ist oder sehr groß ist.[15] Sollte der Gradient klein sein, macht diese Eigenschaft das Lernen für den Algorithmus schwerer und es wird mehr Zeit benötigt, um ähnliche Ergebnisse zu erzielen wie bei anderen Aktivierungsfunktionen. Nachfolgend sind einige gängige Aktivierungsfunktionen aufgelistet.

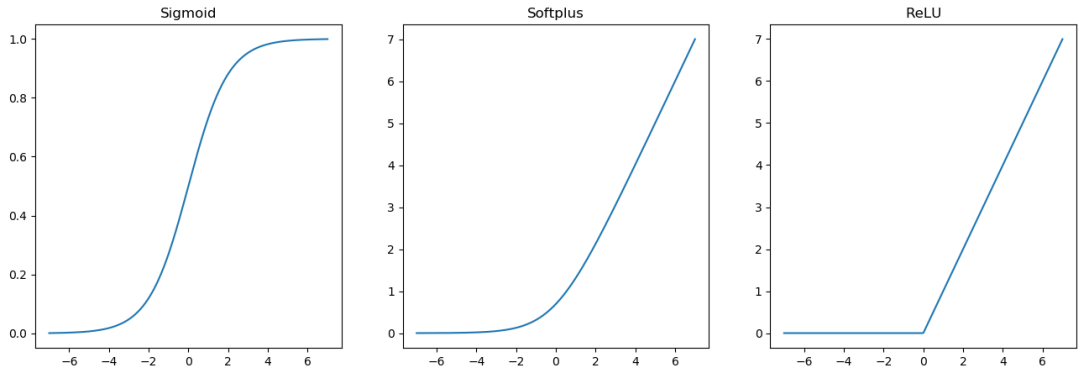


Abbildung 4: Gängige Aktivierungsfunktionen
Quelle: Eigene Darstellung

- Sigmoid: Bei der Sigmoid-Funktion liegt der Wertebereich zwischen 0 und 1. Daher wird diese Funktion häufig bei Klassifizierungsproblemen eingesetzt um zu bestimmen, mit welcher Sicherheit der Algorithmus eine Klassenzugehörigkeit vorhersagt. Eine weitere nützliche Eigenschaft ist die leichte Differenzierbarkeit $\sigma'(z) = \sigma(z)(1 - \sigma(z))$, da die Werte beim maschinellen Lernen oftmals bereits berechnet wurden und sie somit

[13] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 172.
[14] Vgl. Rey/Wender (2011), S. 22 f.
[15] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 179.

wiederverwendet werden können. Ein Nachteil dieser Funktion ist, dass der Gradient bei z≪0 und z≫0 sehr klein ist und das Lernen somit erschwert wird.[16]

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- Softplus: Die Softplus-Funktion hat einen Wertebereich 0 bis ∞. Sie kommt zur Vorhersage von positiven Werten zum Einsatz.

$$\varsigma(x) = \ln(1 + e^{x})$$

- Rectified Linear Unit (ReLU): Die ReLU-Funktion verhält sich ähnlich wie die Softplus-Funktion, allerdings ist der Übergang deutlich härter. Die ReLU-Funktion ist streng genommen nicht differenzierbar bei $x = 0$, jedoch wird sie in der Praxis häufig verwendet, da sie einfach zu implementieren ist und die Steigung konstant bleibt.[17]

$$ReLU(x) = \max(0, x)$$

2.2.2 Konvolutionale Netzwerke

Konvolutionale Netze werden überwiegend bei der Analyse von Bilddaten verwendet. Der Unterschied zu klassischen neuronalen Netzen ist, dass nicht länger ein Vektor an Daten analysiert wird, sondern eine Matrix an Daten, das heißt die Bilddaten stehen in einer Korrelation zueinander. Bildpunkte, die näher beieinander liegen, stehen in einer engeren Verbindung als weit entfernte Bildpunkte. [18] Vergleichbar wie bei einem tiefen neuronalen Netz ist das Netzwerk auch in verschiedene Schichten aufgeteilt. Jede dieser Schichten besteht typischerweise aus folgenden Elementen:

- einer Konvolutionsoperation
- einer Aktivierungsfunktion
- und einer Pooling-Operation

Durch das Verketten dieser Schichten ist es möglich, genau wie bei neuronalen Netzen, komplexere Funktionen zu erlernen, die leistungsfähiger als alleinstehende sind. Die letzten Schichten eines konvolutionalen Netzes sind üblicherweise neuronale Netze, deren Input die Ausgabe der letzten Konvolutionsschicht darstellt, der als Vektor neu arrangiert worden ist.

[16] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 182.
[17] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 192.
[18] Vgl. Bishop (2006), S. 267.

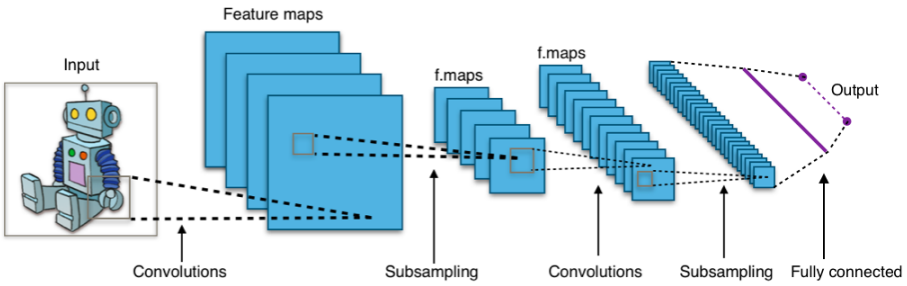


Abbildung 5: Schematischer Aufbau eines konvolutionalen Netzwerkes
Quelle: Aphex34 (2015)

Bei der Konvolutionsoperation wird ein Filter auf einen Teilbereich einer Matrix, bzw. einem Array an Matrizen, angewandt, um einen neuen Wert zu berechnen. Durch das schrittweise Verschieben des Filters, dem sogenannten Stride, wird die ganze Matrix durchlaufen und eine neue Matrix generiert, die komprimierte Informationen des Bildes enthält. Der Filter ist üblicherweise quadratischer Natur und umfasst n^2 Gewichte und einen Bias-Parameter. In der Vergangenheit wurden viele dieser Filter per Hand entworfen, um bestimmte Effekte bei der digitalen Bildverarbeitung zu erzielen, wie einen Glättungseffekt oder Kantenerfassung-Template, um beispielsweise abrupte Änderungen in dem Bild aufzuspüren. Ohne allerdings Filter von Hand zu entwerfen, benutzen wir das Maschinelle Lernen um Filter zu entdecken, die genau für die Aufgabenstellung relevante Informationen extrahieren. Die nachfolgende Abbildung zeigt die Konvolutionsoperation mit einem 3x3 Filter. Die Anzahl der Parameter umfasst somit 9. Hinzukommt noch ein Bias Parameter und somit erhöht sich die Anzahl der Parameter auf 10. Die mathematische Vorschrift um den Wert der neuen Matrix an der Stelle i, j zu berechnen lautet:[19]

$$S(i,j) = (I * K)(i,j) = \sum_n \sum_m I(i+m, j+n) * K(m,n)$$

$S(i,j)$ entspricht den berechneten Wert der generierten Matrix an Stelle $S_{i,j}$.

I entspricht der Input Matrix bzw. der Eingabe der Schicht.

K entspricht dem Filter, der in der englischen Literatur auch als Kernel bezeichnet wird.

i,j bezeichnen die jeweilige Reihe und die jeweilige Spalte, an der die Operation durchgeführt werden soll. Die zuvor festgelegte Schrittweise gibt an, welcher Wert auf i und j hinzuaddiert wird und an welcher Stelle die darauffolgende Operation durchgeführt werden soll.

[19] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 332 f.

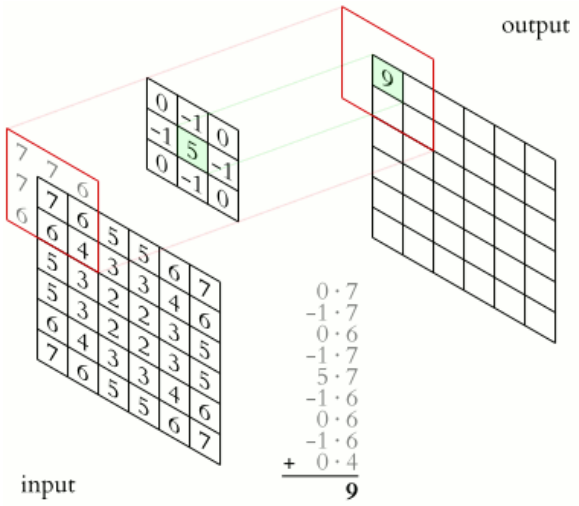


Abbildung 6: Visualisierung der Konvolutionsoperation
Quelle: Michael Plotke (2013)

Würde in der Praxis nur ein Filter pro Inputmatrix verwendet werden, würde es zu einer starken Unteranpassung kommen. Neun Parameter, bzw. 10 Parameter, sind nicht ausreichend genug, um genügend Informationen aus der Matrix zu extrahieren, um eine brauchbare Analyse durchführen zu können. Deshalb werden mehrere Filter gleichzeitig angewendet und die resultierende Matrix wird ein Array an Matrizen mit der Länge entsprechend der Anzahl der Filter. Die erste Matrix, das ursprüngliche Bild, ist häufig auch ein Array an Matrizen da die einzelnen Kanäle für den roten, grünen und blauen Farbton separat gespeichert werden. Im Falle eines Graubildes handelt es sich allerdings um eine einfache Matrix.

Nachdem jede Konvolutionsoperation in der entsprechenden Schicht vollzogen wurde wird auf jeden berechneten Wert noch ein Bias-Parameter hinzuaddiert und eine Aktivierungsfunktion auf die berechneten Werte angewendet.

Eine weitere Operation ist die Pooling-Operation. In ihr wird auf den zuvor berechneten Schritt ein weiterer Filter angewendet, der die Werte gegenüber kleinen Änderungen weniger anfällig machen soll. Die Pooling-Operation erreicht dieses Verhalten dadurch, dass der Filter je nach Wahl den Mittelwert oder den Maximalwert zurückgibt. [20]

2.3 **Das Lernen der Modelle**

Bisher haben wir zwei gängige Algorithmen für das maschinelle Lernen betrachtet. Allerdings sind wir noch nicht auf den charakteristischen Lernprozess eingegangen, der es erlaubt, Wissen in den Parametern abzuspeichern. Wir werden uns in diesem Kapitel erst mit der Performancemessung vertraut machen und anschließend das Konzept der Maximum-Likelihood-Estimation

[20] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 342 f.

kennenlernen. Abschließend werden wir diese dann in dem Kontext des maschinellen Lernens anwenden.

2.3.1 Performancemessungen

In vielen Bereichen des maschinellen Lernens wird den Daten ein bestimmtes statistisches Modell unterstellt, das diese Daten generiert, den sogenannten Daten generierenden Prozess P_{Data}.[21] Die zentrale Überlegung dabei ist, dass wir ein Model entwerfen wollen P_{Model} das sich gleich bzw. ähnlich verhält wie P_{Data}. Ziel der Performancemessung ist es, die Abweichung der Verteilungen, also die Menge an Unsicherheiten, zu quantifizieren. Anders formuliert messen wir die Distanz zwischen der Vorhersage des Datenpunkts und dem tatsächlichen Wert, dem Soll-Wert. In der Literatur finden sich viele Metriken, um die Distanz zwischen zwei Punkten zu bestimmen. Eine dieser Metriken ist beispielsweise die L2-Norm, die auch als „Squared"-Norm bezeichnet wird. Sie entspricht am ehesten unserem natürlichen Verständnis einer Distanz. Es ist die euklidische Norm, die zusätzlich potenziert wird:

$$L_2(x,y) = \left[\sqrt{\sum_i (x_i - y_i)^2}\right]^2 = \sum_i (x_i - y_i)^2$$

Sämtliche Elemente des Differenzvektors zwischen x und y werden potenziert und aufsummiert. In der Literatur wird diese Metrik oft ohne Herleitung eingeführt und dem Leser eine intuitive Erklärung gegeben, also dass kleine Abweichungen durch das Potenzieren noch kleiner werden und das große Fehler deutlich ausschlaggebender sind. Allerdings kann die L2-Norm auch mithilfe obiger Überlegungen über zwei verschiedene Verteilungen, und einer speziellen Metrik hergeleitet werden. Die Herleitung ist deshalb relevant, da sich mit ihr auch andere Metriken herleiten lassen.

Um die Ähnlichkeit zweier statistischer Verteilungen zu bestimmen, kann die Kullback-Leibler-Abweichung (KL-Abweichung, engl. „KL-Divergence") verwendet werden. Sie misst die Ähnlichkeit bzw. die Distanz zweier Distributionen:

$$D_{KL}(P||\,Q) = \mathbb{E}_{x\sim P}\left[\log\frac{P(x)}{Q(x)}\right] = \sum_{x\in X} P(x) * \log\frac{P(x)}{Q(x)}$$

Eigenschaften der KL-Abweichung sind, dass sie keine negativen Werte annehmen kann und dass die Distanz nicht symmetrisch ist, sprich: $D_{KL}(P||Q) \neq D_{KL}(Q||\,P)$. Die KL-Abweichung ist gleich Null, wenn die beiden Distributionen gleich sind.[22]

[21] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 111.
[22] Vgl. MacKay (2005), S. 34.

Die Herleitung erfolgt dabei in Anlehnung an Goodfellow, Bengio und Courville.[23]

Wenn wir für P bzw. Q die beiden Distributionen einsetzen, erhalten wir Folgendes:

$$D_{KL}(P_{Data}||\,P_{Model}) = \mathbb{E}_{x\sim P_{Data}}\left[\log\frac{P_{Data}(x)}{P_{Model}(x)}\right]$$

Um nun die Distanz zwischen den Distributionen zu bestimmen und mit der Annahme das P_{Data} unveränderlich ist, also eine Konstante darstellt, lässt sich Folgendes herleiten:

$$D_{KL}(P_{Data}||\,P_{Model}) = \mathbb{E}_{x\sim P_{Data}}[\log P_{Data}(x) - \log P_{Model}(x)]$$
$$= \mathbb{E}_{x\sim P_{Data}}[\log P_{Data}(x)] - \mathbb{E}_{x\sim P_{Data}}[\log P_{Model}(x)]$$
$$D_{KL}(P_{Data}||\,P_{Model}) \propto -\mathbb{E}_{x\sim P_{Data}}[\log P_{Model}(x)]$$

Der Term auf der rechten Seite wird auch als Cross-Entropy bezeichnet und durch dessen Minimierung nähert sich P_{Model} der Distribution P_{Data} an. Mit anderen Worten verhält sich die Cross-Entropy proportional zu KL-Abweichung zwischen P_{Data} und P_{Model}.

Würde man beispielsweise P_{Model} als eine Normalverteilung betrachten, deren Wert für μ wir in Abhängigkeit von dem Wert für x vorhersagen wollen, würde Folgendes gelten:[24]

$$-\mathbb{E}_{x\sim P_{Data}}[\log P_{Model}(x)] = -\mathbb{E}_{x\sim P_{Data}}[\log P_{Model}(y|\mu,\sigma=1)]\; mit\; \mu = f(x;\theta)$$

‚y' entspricht dabei dem richtigen Wert, der in Relation zu x steht. Je genauer unsere Vorhersage von μ mit gegebenem x ist, desto geringer ist die Cross-Entropy zwischen den Distributionen und unsere Abweichung von P_{Data}. An dieser Stelle lässt sich auch die Überanpassung neu betrachten, da den Wahrscheinlichkeiten von P_{Data} keine Unsicherheiten unterstellt werden (da diese nicht bekannt sind). Dadurch wird P_{Model} versuchen die Menge an Unsicherheit zu reduzieren, obwohl P_{Data} auch Unsicherheiten besitzt durch beispielsweise nicht erfasste weitere Faktoren. Durch einige Umformungen und der Annahme, das σ fix bzw. eine Konstante ist kann die L2-Norm hergeleitet werden:

$$Squared\; Error = |y-\mu|^2\; mit\; \mu = f(x;\theta)$$

Um den Error für den vollständigen Datenbestand zu messen wird der Durchschnitt der Werte berechnet, da wir an dem Erwartungswert interessiert sind. Der Faktor $\frac{1}{m}$ ergibt sich durch die zusätzliche Annahme, dass es sich um eine unabhängige und eine identisch verteilte Distribution handelt (engl. independent and identical distributed – i.i.d.):

$$MSE = -\mathbb{E}_{x\sim P_{Data}}[Squared\; Error] = -\frac{1}{m}\sum_{i=1}^{m}|\,y_i - y_i'|^2\; mit\; y_i' = f(x_i;\theta)$$

Analog lässt sich eine Error Funktion auch für eine bernoullische Verteilung herleiten:

$$Binary\; Cross\; Entropy = \mathrm{y}\log(\phi) + (1-y)\log(1-\phi)\; mit\; \phi = f(x;\theta)$$

[23] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 132.
[24] Vgl. Bishop (2006), S. 29.

Die „Binary Cross Entropy"- Metrik wird bei der Vorhersage von Klassenzugehörigkeiten verwendet allerdings ist eine intuitive Erklärung im Gegensatz zu der L2-Norm deutlich schwerer, doch durch die formale Herleitung sind wir in der Lage, auch für diese Problemstellung eine Metrik aufzustellen.

In der Literatur wird häufig die Error-Funktion mit $\mathcal{L}$ abgekürzt für eine Funktion, die den Abstand misst zwischen dem geschätzten Wert $f(x;\theta)$ und dem tatsächlichen Wert y. Zusammengefasst besteht die Hauptaufgabe der Performancemessung darin, zwischen zwei verschiedenen Algorithmen den besten auswählen zu können. Die Aussagekraft der verschiedenen Metriken ist aber mehr oder weniger abstrakt und gibt nur bedingt Auskunft über die Performance des Algorithmus. Ein Algorithmus könnte trotz eines Fehlerwerts richtige Ergebnisse liefern. Beispielsweise wenn wir einen Klassifizierungsalgorithmus für Spam Mails erstellen wollen würde eine 51% Chance genügen um eine „Spam-Flag" für eine E-Mail zu setzen. Allerdings würden die Performancemessung trotzdem einen Fehlerwert enthalten, da wir eigentlich für diese E-Mail eine hundertprozentige Genauigkeit anstreben. Deswegen ist es in der Praxis häufig sinnvoll alternative Metriken zu erstellen, die eine Aussagekraft besitzen, die für Menschen nachvollziehbar sind. Die Marketing-Abteilung wäre beispielsweise an dem Verhältnis interessiert, das eine E-Mail falsch klassifiziert oder richtig klassifiziert wird.

2.3.2 Maximum-Likelihood-Estimation

Bei der Maximum-Likelihood-Estimation geht es im Prinzip darum, ein Set von Parametern zu finden, sodass die Wahrscheinlichkeit bzw. die Likelihood der Daten maximiert wird. Es wird ein Wert für θ_{ML} geschätzt und anschließend die Likelihood für diesen Wert berechnet. Der Wert für θ, der die Likelihood maximiert, ist die gesuchte Konfiguration für Parameter.

Beispielsweise wenn $\mathbb{X} = \{\mathrm{x}_1, \ldots, \mathrm{x}_m\}$ für eine Reihe von m Datenpunkten steht, sieht die Formel zur Berechnung der Likelihood wie folgt aus:

$$\theta_{ML} = \arg_{\theta} max\, P_{Model}(\mathbb{X};\theta) = \arg_{\theta} max \prod_{i=1}^{m} P_{Model}(x_i;\theta)$$

Da es bei der Berechnung zu einem numerischen Underflow, kommen kann wird anstelle der Likelihood die log-Likelihood maximiert. Dies ist möglich, da der Wert von θ_{ML} bei beiden Funktionen identisch bleibt.[25]

$$\theta_{ML} = \arg_{\theta} max \sum_{i=1}^{m} \log(P_{Model}(x_i;\theta))$$

[25] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 146.

In der Praxis ist es oftmals nicht möglich, sämtliche Werte für θ durchzuprobieren. Daher wird oftmals ein iteratives Verfahren angewendet, bei dem sich die Schätzung von θ schrittweise dem Wert von θ_{ML} annähern soll.

2.3.3 Lernen der Modelle mithilfe des Gradientenabstiegsverfahrens

Da wir uns in den vorherigen Abschnitten mit der Performance des Algorithmus beschäftigt haben, werden wir nun dieses Wissen nutzen um den Algorithmus dementsprechend zu optimieren. Das Verhalten des Algorithmus wird bestimmt durch eine Reihe von veränderbaren Parametern θ. Die geeigneten Werte für θ ergeben sich aus der Optimierung der Performance-Funktion um die log-Likelihood zu maximieren. Eine Möglichkeit um diese schrittweise Optimierung durchzuführen ist das Gradientenabstiegsverfahren.[26] Dazu wird die erste Ableitung berechnet und die Parameter θ so verändert, dass die Error-Funktion reduziert wird. In diesem Fall sind die Inputs $\mathbb{X}$ und die gewünschten Outputs $\mathbb{Y}$ fix und die Funktion $f(x;\theta)$ wird zu einer Funktion von $f(\theta)$. Der neue Wert für θ ergibt sich durch die folgende Formel:

$$\theta \leftarrow \theta + \epsilon * \Delta_\theta \mathcal{L}(\mathbb{X}, \mathbb{Y}, \theta)$$

ϵ entspricht dem so genannten Lernparameter. Er gibt an, wie die Steigung den Parameter θ beeinflussen soll. Er wird meist vorab unveränderlich festgelegt. Die Steigung ist dabei von der Performancefunktion abhängig bzw. dem Deltawert zwischen dem vorhergesagten und dem tatsächlichen Wert mit gegebenen Parametern. Eine Möglichkeit, um die Gradienten von dem neuronalen Netz von den versteckten Schichten zu berechnen, ist der sogenannte Backpropagation-Algorithmus, der das Lernen von tiefen neuronalen Netzen erst möglich macht.

Der Backpropagation-Algorithmus arbeitet dabei vereinfacht ausgedrückt wie ein umgekehrtes vorwärts gerichtetes neuronales Netzwerk. Das heißt, es wird von einem Output zu einem Input propagiert und die Eingabe ist der Deltawert zwischen der observierten und der gewünschten Ausgabe. Dabei ist Backpropagation nur eine Möglichkeit, um die Gradienten zu bestimmen. Die eigentliche Anpassung der Gewichte wird beispielsweise durch das Gradientenabstiegsverfahren vollzogen. Die Berechnung für die Gradienten ist dabei rekursiv, bzw. es wird der Deltawert der Output-Unit berechnet und anschließend schrittweise für die einzelnen versteckten Schichten.

Der Backpropagation-Algorithmus lässt sich dabei nach Mitchell in fünf verschiede Schritte einteilen, die nötig sind, um eine Iteration des Lernprozesses auszuführen. Diese Schritte werden

[26] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 98.

so lange wiederholt, bis der Algorithmus konvergiert oder eine maximale Anzahl von Schritten erreicht wurde.[27]

Die einzelnen Schritte des Backpropagation-Algorithmus sind wie folgt:

1. Zunächst wird eine vorwärts gerichtete Propagation ausgeführt, um die Aktivierungen der einzelnen Schichten zu berechnen und um die Vorhersage des Wertes für y zu erhalten. In der Praxis werden diese Werte zwischengespeichert, um die Werte nicht erneut berechnen zu müssen. Das beschleunigt den Rechenaufwand, da die gleichen Operationen nicht erneut ausgeführt werden müssen. Es ist in diesem Fall sinnvoll, Speicherplatz gegen Rechenzeit einzutauschen.
2. Es werden die Deltawerte für die letzten Neuronen in der letzten Aktivierungsschicht berechnet. Der Wert für $netzinput_i$ ist dabei der berechnete Wert für das Neuron i, bevor eine beliebige Aktivierungsfunktion ϕ angewendet wurde.
$$\delta_i = \phi'(netzinput_i) * (y_i - y_i')$$
3. Berechne rekursiv die Deltawerte für die einzelnen Neuronen in den versteckten Schichten. Dabei iteriert L sämtliche Neuronen, zu denen das Neuron i in Beziehung steht und die Informationen von dem Neuron i ausgehen. W ist ein Ausschnitt der Gewichtsmatrix, der die Beziehung zwischen den Neuronen i und dem Neuron l beschreibt.
$$\delta_i = \phi'(netzinput_i) * \sum_L (\delta_l * w_{li})$$
Der Term $\sum_L(\delta_l * w_{li})$ lässt sich auch als Skalarprodukt zwischen den Deltawerten der Neuronen von der nachfolgenden Schicht und der Gewichtsmatrix auffassen. Visuell entspricht es einem Propagationsschritt allerdings in der entgegengesetzten Richtung.
4. Berechne die Veränderungen der einzelnen Gewichte zwischen den Einheiten i und j:
$$\Delta W_{ij} = \delta_i * a_j$$
a_j entspricht dabei dem Aktivitätslevel des Neurons j.
5. Führe einen Schritt des Gradientenabstiegsverfahren aus und berechne die neuen Elemente der Gewichtsmatrix W:
$$W_{ij} \leftarrow W_{ij} + \epsilon * \Delta W_{ij}$$

Die Herleitung des Backpropagation-Algorithmus zeigt Mitchell dabei anhand der Kettenregel der Differentialrechnung.[28]

Zur Erinnerung: Wir berechnen eine vorläufige Ausgabe mit der Formel $z_i = \sum_l W_{il} * a_l$ und wenden anschließend eine beliebige Aktivierungsfunktion ϕ an und erhalten einen Wert, der

[27] Vgl. Mitchell (1997), S. 98
[28] Vgl. Mitchell (1997), S. 101 ff.

für die neue Schicht weitergereicht wird, $a_l = \phi(z_l)$. Als Performancefunktion und zur Bestimmung des Fehlerterms benutzen wir die „Squared Error"-Funktion (E) für ein einziges Trainingsbeispiel.

Wir suchen die Ableitung für die jeweiligen Gewichte W_{il}. Formal ausgedrückt:

$$\frac{\delta E}{\delta W_{il}} = \frac{\delta E}{\delta z_i} * \frac{\delta z_i}{\delta W_{il}} = \delta_i * a_l$$

Der Wert für $\frac{\delta E}{\delta z_i} = \delta_i$ ist zu diesem Zeitpunkt unbekannt und stellt eine gesuchte Größe dar. Die Ableitung für $\frac{\delta z_i}{\delta W_{il}} = a_l$ lässt sich anhand obiger Formel ablesen. Wie bereits oben beschrieben ist der Wert für a_l der zwischengespeicherte Wert, der bei der vorwärts gerichteten Propagation berechnet wurde.

Für $\delta_i = \frac{\delta E}{\delta z_i}$ existieren zwei verschiedene Fälle. Entweder handelt es sich um ein Ausgabeneuron oder ein Neuron innerhalb einer verdeckten Schicht. Im ersten Fall handelt es sich um ein Ausgabeneuron und die Vorhersage lässt sich anhand dieses Neurons ablesen. Wir gehen vereinfachend davon aus, dass es sich um eine lineare Aktivierung handelt.

Es wird der Deltawert δ_k wie folgt berechnet:

$$\delta_k = \frac{\delta E}{\delta z_k} = \frac{\delta E}{\delta y'} * \frac{\delta y'}{\delta z_k} = \frac{\delta E}{\delta z_k} = -2 * (y - y') * \frac{\delta y'}{\delta z_k} = -2 * (y - y')$$

Im zweiten Fall handelt es sich um ein Neuron in einer versteckten Schicht und wir können auf einen Deltawert δ_j der nachfolgenden Schichten zugreifen. Wie bereits oben beschrieben berechnen wir bei Backpropagation in entgegengesetzter Richtung. Es ist also zwingend erforderlich, δ_j zu berechnen bevor wir δ_i berechnen können.

Die Deltawerte der verdeckten Schichten lassen sich wie folgt berechnen:

$$\delta_i = \frac{\delta E}{\delta z_i} = \sum_j \frac{\delta E}{\delta z_j} * \frac{\delta z_j}{\delta z_i} = \sum_j \delta_j * \frac{\delta z_j}{\delta z_i}$$

$$\frac{\delta z_j}{\delta z_i} = \frac{\delta z_j}{\delta a_i} * \frac{\delta a_i}{\delta z_i} = W_{ji} * \phi'(z_i)$$

Werden die einzelnen Formeln ineinander eingesetzt, entstehen die Formeln der jeweiligen Schritte wie weiter oben beschrieben.

Prinzipiell lässt sich der Backpropagation-Algorithmus auf eine Vielzahl an Modellen anwenden um die Gradienten zu bestimmen. Es bietet sich dabei an einen, sogenannten Berechnungsgraphen zu definieren, in dem die einzelnen mathematischen Operationen als Knotenpunkte aufgefasst werden, der die Beziehungen zwischen anderen Elementen mithilfe von Kanten beschreibt.[29] Aufgrund dieser Flexibilität für verschiedene Problemstellungen und der

[29] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 205.

Einsatzmöglichkeiten für verschiedene Modelle bieten entsprechende Frameworks für das maschinelle Lernen, wie beispielsweise TensorFlow, die Möglichkeit, an einen Graphen zu definieren, der automatisch diese Formeln berechnet.

Das Lernen für neuronale Netze erweist sich in der Praxis als problematisch, da das Gradientenabstiegsverfahren ein lokales und kein globales Minimum anstrebt. Ein Forschungsgebiet des maschinellen Lernens beschäftigt sich mit dem Entwickeln von Verfahren, die diese lokalen Minima überspringen und hoffentlich bessere lokale Minima finden. Häufig eingesetzte Verfahren in der Praxis sind beispielsweise Lernparameter die sich mit der Zeit verringern oder auch auch Verfahren die das Momentum des Gradienten nutzen oder auch Verfahren den einzelnen Neuronen eine statistische Verteilung unterstellt und die Aktivierungen dementsprechend anpasst.

Graphisch kann das Gradientenabstiegsverfahren folgendermaßen interpretiert werden:

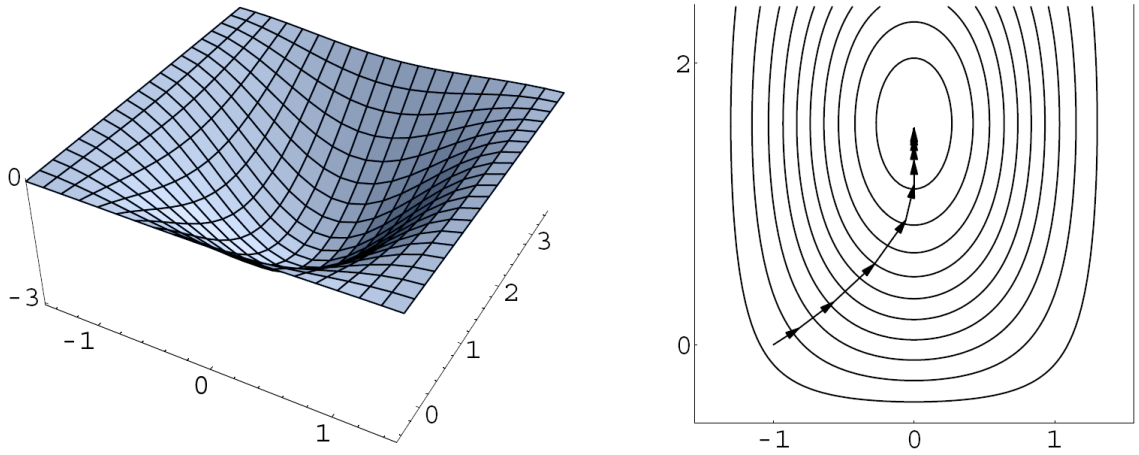


Abbildung 7: Gradientenabstiegsverfahren
Quelle: Kriesel (2007), S. 62.

Die Oberfläche zeigt die Bereiche, an denen die Performance-Funktion groß bzw. klein ist. Der schwarze Strich zeigt die einzelnen Stationen des Gradientenabstiegsverfahrens an bzw. die Verschiebung der Position durch die Anpassung von den Parametern θ. In dieser Abbildung wird der Algorithmus lediglich von zwei Parametern charakterisiert um die Vorgehensweise des Gradientenabstiegsverfahrens zu veranschaulichen. In der Praxis bestehen die Algorithmen aus deutlich mehr Parametern und die Performancefunktion bietet eine Vielzahl an lokalen Optima. In der Regel handelt es sich um ein nicht konvexes Optimierungsproblem. Die Gründe dafür sind im Fall von neuronalen Netzen vielfältig. Beispielsweise durch eine sogenannte Gewichts-Symmetrie, weil die einzelnen Gewichte der Neuronen innerhalb einer Schicht beliebig ausgetauscht werden können, da die Reihenfolge der Neuronen keine Rolle spielt. Ein weiteres Optimierungsproblem sind Sattelpunkte. Mathematisch lässt sich zeigen, dass das Verhältnis zwischen Sattelpunkten und lokalen Optima in einem hochdimensionalen Raum exponentiell wächst. Es ist wahrscheinlicher an einen Sattelpunkt zu geraten als einem lokalen Optimum. Prinzipiell sind

Sattelpunkte schlecht, da die Gradienten immer kleiner werden, welches den Trainingsprozess verlangsamt, allerdings scheint das Gradientenabstiegsverfahren, empirisch betrachtet, die Fähigkeit zu besitzen diese Sattelpunkte zu überwinden.[30]

Die Oberfläche der Performancefunktion zu analysieren ist ein Forschungsgebiet im Bereich des maschinellen Lernens, in welchem noch aktiv Forschung betrieben wird. Ein besseres Verständnis dieser erlaubt das bessere Designen der Architektur, dem Entwickeln neuer Aktivierungsfunktionen und der Erstellung von Verfahren, die schneller konvergieren u.v.m.

[30] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 284 ff.

3 Inferenz

Unter Inferenz (lat. „inferre“) wird das Schlussfolgern von Sachverhalten verstanden. Im Wesentlichen geht es darum, zu überprüfen, welche Attribute gegeben sind und welche Informationen sich aus diesen Attributen herleiten lassen. Beispielsweise wenn wir wissen, dass jeder lebende Mensch atmet und wenn wir wissen, dass Sokrates lebt sowie ein Mensch ist, wäre eine Schlussfolgerung, dass Sokrates atmen würde. Diese spezielle Form der Schlussfolgerung wird auch Implikation genannt. Die Mathematik ist aufgebaut auf dem Schlussfolgern von Grundannahmen bzw. Axiomen, um neues Wissen zu generieren, das zum weiteren Schlussfolgern verwendet wird. Im Bereich der Mustererkennung und des unüberwachten Lernens wird der Inferenzbegriff auf eine ähnliche Weise verwendet. Es geht darum, Datensätze zu analysieren um Informationen, zu finden die für die weitere Verwendung nützlich sind. Beispielsweise: Ist der Kunde kreditwürdig? In wie viele Zielgruppen lassen sich unsere Kunden einteilen? Zu welcher Zielgruppe gehört der neue Kunde?

Diese erste Art der Fragestellung lässt sich durch das Observieren von vergangenen Ereignissen beantworten. Es lässt die sich Wahrscheinlichkeit ermitteln, mit der die Aussage zutrifft bzw. nicht zutrifft. Die zweite Art der Fragestellung befasst sich mit dem Erfassen einer verdeckten Struktur. Gibt es Gemeinsamkeiten zwischen unseren Kunden und welche wären das? Die dritte Fragestellung ist ähnlich unserer Problemstellung des vergangenen Kapitels. Allerdings ist diese Zielgruppe unbekannt bzw. mit der Zugehörigkeit des Kunden zur Gruppe verändert sich die Gruppe. Jedoch kann jede dieser Fragestellungen als Inferenzproblem aufgefasst werden.

In diesem Kapitel werden wir uns überwiegend mit variational inference beschäftigen, die auf diese Art von Problemstellungen angewendet werden kann. Wir werden uns zunächst mit dem Bayes-Theorem vertraut machen, mit dem eine Inferenz durchgeführt werden kann sowie dem Problem bei dieser Methode. Anschließend werden wir im Detail den Expectation-Maximization-Algorithmus, der das Fundament für variational inference bildet, mit der wir dieses Kapitel abschließen.

3.1 **Bayes Theorem**

Das Bayes-Theorem spielt eine zentrale Rolle im Bereich der Musterkennung. Es wird auch häufig als „umgekehrte Wahrscheinlichkeit“ bezeichnet. In dem vergangenen Kapitel konnten wir beispielsweise die bedingte Wahrscheinlichkeit von y mit einem gegebenen Wert x berechnen. Das Bayes Theorem dreht nun die Fragestellung um und fragt: „Was ist die bedingte

Wahrscheinlichkeit von x mit einem gegebenen y?“ Die Bayes-Formel lässt sich durch die Umkehrung der Produktregel ($P(A,B) = P(A|B)P(B)$) einfach zeigen:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

In der Literatur haben die einzelnen Formelelemente bestimmte Bezeichnungen. Der Term $P(A|B)$ wird beispielsweise diesem Zusammenhang als Posterior bezeichnet. $P(B|A)$ ist die Likelihood, die mit dem Prior $P(A)$ multipliziert wird und anschließend durch die Evidence $P(B)$ geteilt wird. Die Evidence fungiert als Normalisierungskonstante und ist die Summe über sämtliche Konfigurationen von A. Sie dient dazu valide Wahrscheinlichkeitswerte zu erhalten.

Die Evidence zu berechnen kann sich in der Praxis als zu komplex erweisen. Beispielsweise wenn wir erfassen möchten, ob es sich bei dem Bild B um eine Katze A handelt, wäre die gegensätzliche Fragestellung, ob es sich bei der Katze A um Bild B handelt. Diese Fragestellung ist deutlich kniffeliger da A prinzipiell auf viele Bilder zeigt, die diese Eigenschaft erfüllen. Die Dimensionsunterschiede sind entscheidend da die Evidence über sämtliche Konfigurationen berechnet wird und oftmals ist die Vertauschung der Fragestellung ein guter erster Schritt zur Lösung des Problems, je nachdem, ob das eine oder andere einfacher zu berechnen ist.

Eine Möglichkeit, um eine Intuition über das Bayes-Theorem zu erhalten, ist durch das Studieren eines Beispiels in Anlehnung an MacKay. [31] Angenommen, wir wären von unserem Arzt auf eine gefährliche Krankheit getestet worden und dieser Test wäre positiv ausgefallen. Der Arzt gibt den Hinweis, dass die Genauigkeit des Tests bei 99% liegt. Wie besorgt wären wir? Anhand von Statistiken erfahren wir das ca. 1 von 1000 Menschen betroffen sind. Eingesetzt in die Formel ergibt sich:

$$P(Krank|Test_P) = \frac{P(Test_P|Krank)P(Krank)}{P(Test_P|-Krank)P(-Krank) + P(Test_P|Krank)P(Krank)}$$

$$= \frac{0.99 * 0.001}{0.99 * 0.001 + 0.01 * 0.999} \approx 9.01\%$$

Laut Bayes Theorem besteht eine 9.01% Chance, dass wir eine ernsthafte Krankheit haben, im Verhältnis zu den 99% eine deutliche Besserung. Der Schlüssel dieser Verzerrung liegt bei der zusätzlichen Information zu der Auftretungswahrscheinlichkeit. Beispielsweise haben wir bei obigen Zahlen circa zehn Falschklassifizierungen. Die Chance, dass wir die Falschklassifizierung sind oder der Test wahrhaftig positiv ist, drückt das Bayes-Theorem in diesem Beispiel aus.

[31] Vgl. MacKay (2005), S. 25.

3.2 **Inferenzmethoden**

In dem vergangenen Beispiel konnten wir eine Inferenz durchführen, indem wir durch sämtliche verschiedene Fälle iteriert haben. In diesem Fall wird von einer vollständigen Enumeration gesprochen. Allerdings existieren noch andere Methoden, um Inferenzen durchzuführen. [32]
In der Literatur haben sich folgende Verfahren etabliert:

- Komplette Enumeration
- Laplace-Methoden
- Monte Carlo-Methoden
- Variational inference

Wir werden uns nur mit der kompletten Enumeration sowie mit variational inference befassen. Es sollte allerdings erwähnt werden, dass auch andere Methoden existieren, die entsprechende Vor- bzw. Nachteile haben, um das Inferenz-problem zu lösen.
Die komplette Enumeration bezieht sich darauf, dass wir sämtliche Teile der Gleichung des Bayes-Theorems haben, oder sich diese Berechnung durchführen lässt. Der Vorteil ist, dass wir exakte Ergebnisse erhalten, allerdings durch einen oftmals sehr hohen Rechenaufwand. Dieser Rechenaufwand ist teilweise so hoch, dass er in einer nicht annehmbaren Zeit durchführbar ist. Der Grund ist häufig die Evidence, da sie über sämtliche Konfigurationen berechnet werden muss. Das heißt, dass alle Faktoren ausgegrenzt werden müssen, um die Evidence exakt berechnen zu können. In dem Fall von diskreten Variablen bedeutet dies eine Reihe von Summen, um sämtliche verschiedene Features miteinander zu kombinieren.

$$P(B) = \sum_{A} P(B, A) = \sum_{A_1} \sum_{A_2} \dots \sum_{A_m} P(B, A_1, \dots A_m)$$

Die Rechendauer wächst exponentiell, da für jede weitere Dimension für A die „for-Schleife“ erneut durchlaufen werden muss. Eine Folgerung ist, dass sich die Evidence nicht mehr berechnen lässt und sich keine exakte Inferenz mehr durchführen lässt. Dieses Problem wird auch als der Fluch der Dimensionen bezeichnet, der sich auch auf die Anzahl der Daten bezieht, die nötig sind, um eine möglichst genaue Inferenz durchzuführen.[33]
Eine Möglichkeit, um dieses Problem zu lösen, ist die Anwendung sogenannter approximierter Methoden. Variational inference gehört zu dieser Rubrik und wir werden diese anhand des Expectation-Maximization-Algorithmus herleiten.

[32] Vgl. MacKay (2005), S. 294.
[33] Vgl. Bishop (2006), S. 33 ff.

3.3 Expectation-Maximization-Algorithmus

Der Expectation-Maximization-Algorithmus (EM-Algorithmus) ist ein Algorithmus, der sehr häufig im Bereich der Musterkennung und im Data-Mining verwendet wird. Er gilt als Grundlage für variational bayesian inference (VI). Der EM Algorithmus lässt sich vereinfacht ausgedrückt als Zyklus auffassen, in dem zwei Parameter gesucht sind. Durch das abwechselnde Fixieren und Anpassen der Parameter konvergiert der Algorithmus zu einem lokalen Optimum. Wir werden zwei verschiedene Einsatzmöglichkeiten betrachten. Zunächst der k-means-Algorithmus und anschließend den EM-Algorithmus für eine Gaußsche Mischverteilung. Abschließend werden wir das Framework für VI-Methoden herleiten.

3.3.1 K-means als EM-Algorithmus

In der Literatur wird der Algorithmus gerne im Zusammenhang mit dem k-means-Algorithmus eingeführt, bei dem schrittweise die Clusterzugehörigkeiten ermittelt werden und die Mittelpunkte verschoben werden. Dieser Prozess wiederholt sich so lange, bis keine neuen Clusterzuweisungen mehr durchgeführt werden. Diese zwei Schritte können als ein Erwartungsschritt und ein Optimierungsschritt aufgefasst werden. [34]

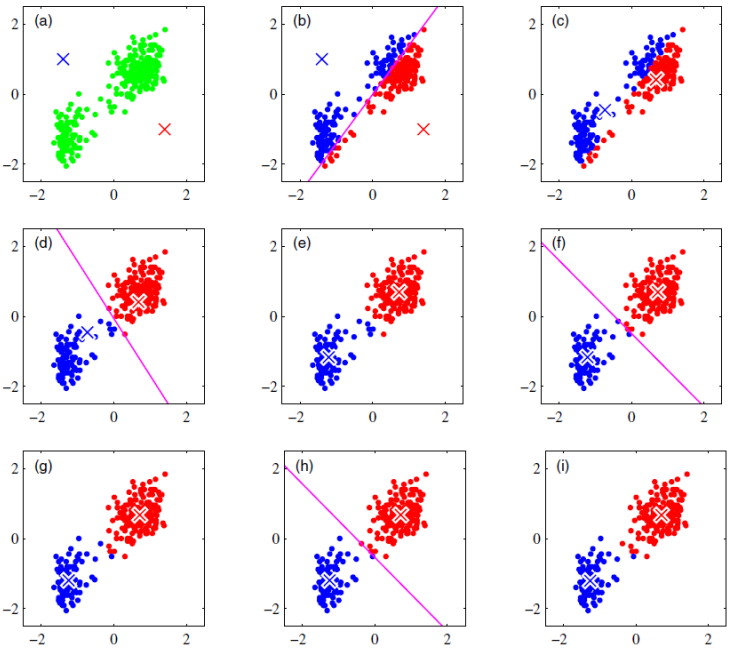


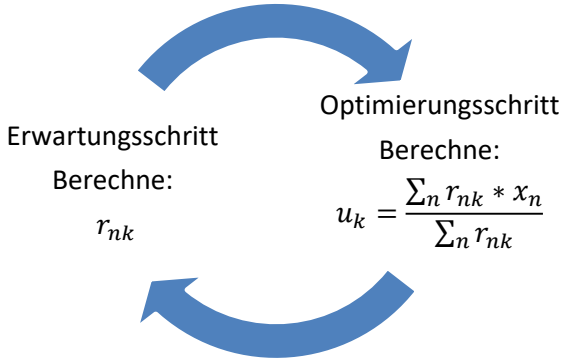
Abbildung 8: K-Means Algorithmus
Quelle: Bishop (2006), S. 426.

Die Abbildung zeigt die einzelnen Iterationen des EM-Algorithmus mit dem Erwartungs- und Optimierungsschritt. Es ist sehr gut erkennbar, dass sich die geschätzten Clustermittelpunkte

[34] Vgl. Bishop (2006), S. 424 ff.

immer weiter den wahren Clustermittelpunkten annähern. Formal haben wir eine Performancefunktion, die den Abstand der N-Datensätze zu dem zugeordneten Cluster misst und eine Anzahl an K Clustern, deren Mittelwert μ_k den Clusterursprung darstellt. Die Zugehörigkeit sowie die Clustermittelpunkte sind vorab unbekannt und stellen die gesuchten Werte des Algorithmus dar. Formal lässt sich das folgendermaßen ausdrücken (dabei stellt r_{nk} die Zugehörigkeit von dem Datensatz n zu dem Cluster k dar):[35]

$$r_{nk} = \begin{cases} 1\ wenn\ k = \arg\min_j |x_n - \mu_j|^2 \\ 0\ andernfalls \end{cases}$$



Der Erwartungsschritt ist die Berechnung der Clusterzugehörigkeit und der Optimierungsschritt ist die Verschiebung der Clustermittelpunkte bzw. die Berechnung des Durchschnitts (Mean) der Datensätze. Es ist besonders hervorzuheben, dass der EM-Algorithmus und VI-Methoden von der Initialisierung der Startpunkte abhängen. Beispielsweise könnte bei einer schlechten Initialisierung ein Cluster statistische Ausreißer darstellen, wohingegen die anderen Cluster die gesamten üblichen Werte abdecken. Dadurch, dass die Initialisierung Einfluss auf das Resultat nimmt, bedeutet dies, dass nicht ein globales, sondern ein lokales Optimum gefunden wurde. In der Praxis ist es üblich, mehrere Durchläufe mit unterschiedlichen Initialisierungen durchzuführen. Die Ergebnisse werden mithilfe einer Performancefunktion, wie die Summe der Distanzen zu den jeweiligen Clustern, miteinander verglichen und das beste Clustering ausgewählt.

3.3.2 EM-Algorithmus für ein mehrdimensionales Verteilungsmodell

Das gleiche Prinzip lässt sich auch für eine Gaußsche Mischverteilung anwenden, bei der die einzelnen Verteilungen verantwortlich für die Generierung des jeweiligen Datenpunkts sind.[36] Dabei wird anstelle einer harten Clusterzuweisung eine weiche Clusterzuweisung gefunden, die

[35] Vgl. Bishop (2006), S. 425.
[36] Vgl. Bishop (2006), S. 439 f.

eine Wahrscheinlichkeit angibt, mit dem der Datenpunkt zu dieser Verteilung gehört bzw. generiert wurde. Im Gegensatz zu k-means sind die einzelnen r_{nk} Wahrscheinlichkeiten.

$$r_{nk} = P(k|x_n) = \frac{P(x_n,k)}{P(x_n)} = \frac{P(k)N(x_n|\mu_k,\Sigma_k)}{\sum_{i=1}^{K} P(k_i)N(x_n|\mu_i,\Sigma_i)} \; mit \; P(k) = \frac{N_k}{N} = \frac{\sum_{n=1}^{N} r_{nk}}{N}$$

In jeder Iteration des Zyklus werden μ und Σ für jeden Cluster erneut berechnet und die Verteilungen verschieben sich entsprechend. Eine wesentliche Änderung zu dem k-means-Algorithmus ist, dass die Kovarianzmatrix Σ verwendet wird, mit der es ermöglicht wird, komplexere Clusterstrukturen zu finden. Ein Problem des k-means-Algorithmus ist, dass die Distanzen zwischen den einzelnen Attributen mit dem gleichen Gewicht in die Berechnung eingehen. Die Gaußsche Mischverteilung in Verbindung mit dem EM-Algorithmus löst elegant dieses Problem, indem die Wahrscheinlichkeit, dass dieser Datenpunkt zu dem Cluster gehört, in die Berechnung einkalkuliert wird. Die neuen Werte für μ und Σ werden wie folgt berechnet:

$$\mu_k' = \frac{\sum_n^N r_{nk} * x_n}{\sum_n^N r_{nk}}$$

$$\Sigma_{k_{jk}}' = \frac{1}{N_k} \sum_{i=1}^{N} r_{nk} * \left(x_j^{(i)} - \mu_{k_j}\right)\left(x_k^{(i)} - \mu_{k_k}\right)$$

μ_k ist der Mittelwert von dem Cluster k und μ_{k_i} ist das i'te Element von dem Vektor μ_k. Analog ist $x_j^{(i)}$ von dem i'ten Datensatz das Attribut j.

Die folgende Abbildung zeigt ein beispielhaftes Clustering mit zwei Gaußschen Mischverteilungen.

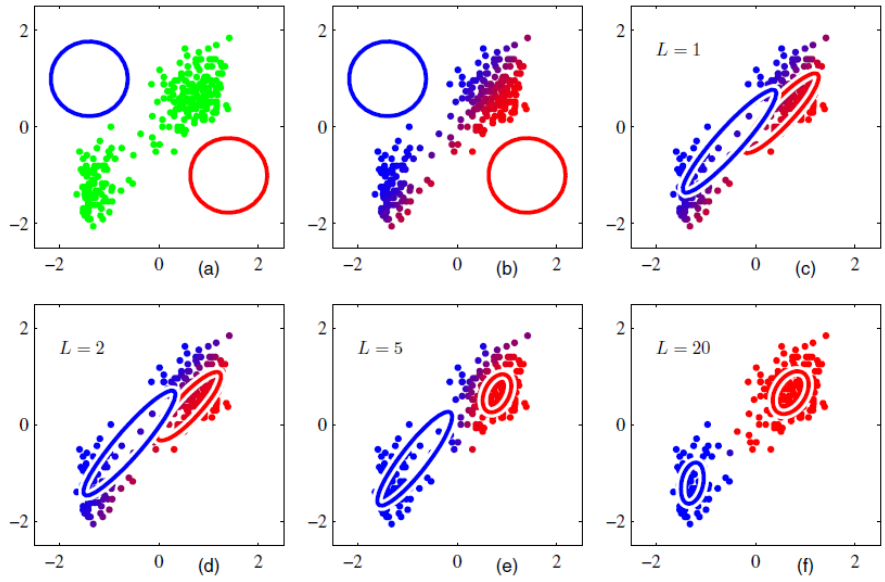


Abbildung 9: Clustering mit einer Gaußschen Mischverteilung
Quelle: Bishop (2006), S. 437.

3.3.3 Herleitung des Variational Inference Frameworks

Wie anfangs beschrieben stellt der Expectation-Maximization-Algorithmus das Herz von variational inference Methoden dar. Wir wollen daher noch das generelle Konzept betrachten sowie die Herleitung nach Blei, Kucukelbir und McAuliffe.[37]

Für die Herleitung führen wir eine Funktion Dichtefunktion $q(z)$ ein, die sich dem wahren optimalen Posterior $P(z|x)$ annähern soll. Diese Dichtefunktion stammt aus einer Familie von Dichtefunktionen D, die sich als der Suchraum herausstellen, und wird vorab festgelegt. Wie wir sehen werden, bildet $q(z)$ eine untere Schranke für den wahren Posterior $P(z|x)$. Um die Abweichung zwischen den beiden Distributionen zu ermitteln, benutzen wir die Kullback-Leibler-Abweichung (KL). Für sie gilt $KL \geq 0$ und die KL-Abweichung ist dann und nur dann gleich Null, wenn die beiden Distributionen gleich sind.

$$q^*(z) = \arg_{q(z)\in D} \min KL(q(z)||\ P(z|x))$$

Per Definition der KL-Abweichung ergibt sich:

$$KL(q(z)||\ P(z|x)) = \mathbb{E}\left[\log\left(\frac{q(z)}{P(z|x)}\right)\right] = \ \mathbb{E}[\log(\mathrm{q(z)})] - \mathbb{E}[\log \mathrm{P(z|x)}]$$

Durch Umformung der konditionalen Wahrscheinlichkeit und der Tatsache, dass $\mathbb{E}[\log(\mathrm{P(x)}] = \log P(x)$ ist, da sie unabhängig von dem Erwartungswert von z ist, ergibt sich:

$$KL\big(q(z)||P(z|x)\big) = \ \mathbb{E}[\log(\mathrm{q(z)})] - \mathbb{E}\left[\log\left(\frac{P(z,x)}{P(x)}\right)\right]$$

$$= \ \mathbb{E}[\log(\mathrm{q(z)})] - \mathbb{E}[\log(\mathrm{P(z,x)})] + \log P(x)$$

Es ergibt sich folgende Gleichung:

$$KL(q(z)||\ P(z|x)) = \ \mathbb{E}[\log(\mathrm{q(z)})] - \mathbb{E}[\log \mathrm{P(z,x)}] + \log P(x)$$

$$\log P(x) = \mathbb{E}[\log \mathrm{P(z,x)}] - \mathbb{E}[\log(\mathrm{q(z)})] + \ KL(q(z)||\ P(z|x))$$

Unser Ziel ist es, den Term $\log P(x) = \log(\sum_z P(x,z))$ zu maximieren. Dadurch, dass es sich um eine Summe innerhalb eines Logarithmus handelt, kann die Maximum-Likelihood-Estimation nicht exakt durchgeführt werden und wir haben keinen Zugriff auf eine geschlossene Lösung mehr.

Wir können allerdings die beiden anderen Terme benutzen, um $\log P(x)$ trotzdem zu optimieren. Sie bilden den sogenannten „Evidence Lower Bound“ (ELBO) und die KL-Abweichung zwischen unserer Zielfunktion $\mathrm{q(z)}$ und $P(z|x)$. Da die KL-Abweichung nicht negativ sein kann und $\log P(x)$ eine Konstante darstellt, gilt $\mathrm{ELBO} \leq \log P(x)$. Wie gut unsere Approximation ist, lässt sich also anhand der Formel ablesen:

$$\log P(x) = ELBO - KL$$

[37] Vgl. Blei/Kucukelbir/McAuliffe (2018), S. 6 f.

Durch Maximierung des ELBO wird die KL-Abweichung immer geringer und nähert sich immer weiter dem Wert von $\log P(x)$ an. Der ELBO dient in dem Sinne als Proxy für unsere eigene Optimierungsfunktion. Durch weitere Umformungen entsteht:

$$ELBO = \mathbb{E}[\log \mathrm{P(z,x)}] - \mathbb{E}\big[\log\big(\mathrm{q(z)}\big)\big] = \mathbb{E}\left[\log\left(\frac{P(z,x)}{q(z)}\right)\right]$$
$$= \mathbb{E}[\log(\mathrm{P(z)}] + \mathbb{E}[\log \mathrm{P(x|z)}] - \mathbb{E}\big[\log\big(\mathrm{q(z)}\big)\big]$$
$$= \mathbb{E}[\log(\mathrm{P(x|z)}] - KL(q(z)||\,P(z))$$

Es ergibt sich die log-Likelihood von $P(x|z)$ und die KL-Abweichung von unserem gewählten Prior und dem tatsächlichen Prior. Daran lässt sich schon ein Problem festmachen, nämlich dass sich unser Suchraum nur auf eine Dichtefunktionsfamilie beschränkt. Eine folge dessen ist, dass wir in der Praxis meist nie exakte Ergebnisse erhalten werden. Die KL-Abweichung bestraft in dem Sinne unsere ursprüngliche Annahme, mit der wir die Daten abbilden wollen, wenn sich diese Daten nicht, oder nicht gut, mit der Annahme abbilden lassen. Es ist in dem Sinne eine Beschränkung, dass wir verletzen können allerdings nur zu einem Preis.

Die Parallele zu dem EM-Algorithmus ist, nach Bishop, dass wir zwei Parameter haben.[38] Einen neuen gesuchten Parameter θ^{new} und einen alten Parameter θ^{old} die unsere beiden Dichtefunktionen beschreiben. Wenn wir für $\mathrm{q(z)} = \mathrm{P(z|x,\theta^{old})}$ einsetzen bzw. unserem aktuell besten Modell können wir auf folgende Gleichung kommen:

$$ELBO = \mathbb{E}\left[\log\left(\frac{P(z,x)}{q(z)}\right)\right] = \sum_z q(z) * \log\frac{P(z,x)}{q(z)} = \sum_z \mathrm{P\big(z\big|x,\theta^{old}\big)} * \log\frac{P(z,x)}{\mathrm{P\big(z\big|x,\theta^{old}\big)}}$$
$$= \sum_z \mathrm{P\big(z\big|x,\theta^{old}\big)} * \log P(z,x) - \sum_z \mathrm{P\big(z\big|x,\theta^{old}\big)} * \log \mathrm{P\big(z\big|x,\theta^{old}\big)}$$
$$= \sum_z \mathrm{P\big(z\big|x,\theta^{old}\big)} * \log P(z,x) + Const$$

Wenn wir für die gemeinsame Wahrscheinlichkeit einen neuen Parameter θ einführen, können wir eine neue Funktion erstellen die sich wie folgt ergibt:

$$Q(\theta,\theta^{old}) = \sum_z \mathrm{P\big(z\big|x,\theta^{old}\big)} * \log P(z,x|\theta)$$

Für diese neue Funktion können wir die Maximum-Likelihood-Estimation anwenden und den neuen Parameter θ^{new} bestimmen, der im nächsten Zyklus den Wert für θ^{old} darstellt.

$$\theta^{new} = \arg_\theta \max Q(\theta,\theta^{old})$$

Die Optimierung erfolgt dabei über $\log P(z,x|\theta)$. Es lässt sich zeigen, dass wenn dieser Term nach dem entsprechenden Parameter abgeleitet wird und die Nullstelle berechnet wird, die Formeln aus dem vorherigen Kapitel entstehen. Im Falle von einer Normalverteilung handelt es sich

[38] Vgl. Bishop (2006), S. 451 f.

um eine Parabel, die nach unten geöffnet ist. Es existiert somit eine Extremstelle, die das Optimum darstellt. Die neue Parameterkonfiguration für θ stellt in dem neuen Zyklus den Wert für θ^{old} da. Die Funktion $Q(\theta, \theta^{old})$ lässt sich also so interpretieren, dass die Funktion die jeweils letzten Elemente der Reihe $\theta_1, \theta_2, \dots, \theta_n$ enthält.

Grafisch kann der EM-Algorithmus folgendermaßen aufgefasst werden:

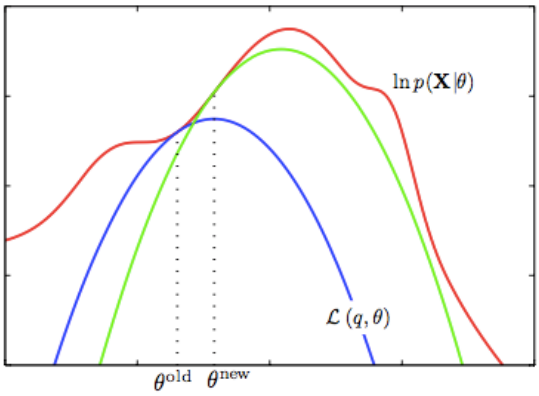


Abbildung 10: Variational Inference
Quelle: Bishop (2006), S. 453.

3.4 **Variational Inference**

Bisher haben wir uns mit dem Expectation-Maximization-Algorithmus vertraut gemacht sowie dessen Herleitung. Wir haben für unsere Funktion mit dem Posterior gleichgesetzt $\mathrm{q(z)} = \mathrm{P(z|x, \theta^{old})}$, um das generelle Framework der Variational Inferenz kennenzulernen. Es ist allerdings nicht zwingend notwendig, diesem Ansatz zu folgen um das Inferenzproblem zu lösen. Wir wollen $q(z)$ so wählen, dass sich die gemeinsame Wahrscheinlichkeit $\log P(z, x)$ möglichst einfach berechnen lässt, allerdings sollte die Funktion flexibel genug sein, um eben möglichst genau zu approximieren. Häufig wird $q(z)$ dabei aus einer Familie von Dichtefunktionen ausgewählt, deren Elemente disjunkt sind bzw. die folgendermaßen beschränkt ist:

$$q(z) = \prod_i q_i(z_i)$$

Mit anderen Worten, die Elemente sollen unabhängig voneinander sein. Jede latente Variable wird von einer eigenen separaten Distribution abgedeckt, die untereinander keine Abhängigkeiten haben. Dieser Ansatz wird auch als „Mean field"-Ansatz bezeichnet und gehört zu den einfacheren Varianten von variational inference. Wissenschaftler haben auch andere Dichtefunktionsfamilien untersucht, die beispielsweise Abhängigkeiten untereinander aufweisen. Dieser Ansatz wird auch als strukturierte variational Inferenz bezeichnet.[39] Dieser Ansatz würde allerdings

[39] Vgl. Blei/Kucukelbir/McAuliffe (2018), S. 8.

an dieser Stelle zu weit führen. Das Modell kann aufgrund einer zu flexiblen Distribution nicht an einer Überanpassung leiden. Die Approximation kann dadurch nur noch genauer werden.[40]
Wie wir gesehen haben, geht es bei VI darum, den ELBO zu maximieren. Wenn wir in die Gleichung $q(z) = \prod_i q_i(z_i)$ einsetzen, erhalten wir Folgendes:

$$ELBO(q) = \mathbb{E}[\log \mathrm{P(z,x)}] - \mathbb{E}[\log \mathrm{q(z)}] = \mathbb{E}[\log \mathrm{P(z,x)} - \log q(z)]$$

$$= \sum_z \prod_i q_i \left[\log \mathrm{P(z,x)} - \log\left(\prod_i q_i\right)\right] = \sum_z \prod_i q_i \left[\log \mathrm{P(z,x)} - \sum_i \log q_i\right]$$

Ein großer Vorteil, wenn die einzelnen Distributionen unabhängig sind, ist, dass die Gleichungen nach genau dieser Distribution umgestellt werden können. Die folgenden Gleichungen isolieren eine einzelne Distribution q_j, die stellvertretend für eine Komponente steht. Nach jeweils einer dieser Komponente wird zyklisch optimiert. Bei der Notation steht $\Sigma_{z_{i\neq j}}$ für eine Reihe über Summen bzw. $\Sigma_{z_1} \dots \Sigma_{z_m}$ jedoch ohne Σ_{z_j}. Die Anzahl der latenten Variablen bestimmt die Anzahl der Sigmas und die Anzahl der Distributionen.[41]

$$= \sum_{z_j} q_j \left[\sum_{z_{i\neq j}} \log \mathrm{P(z,x)} \prod_{i\neq j} q_i\right] - \sum_z \prod_i q_i \sum_i \log q_i$$

$$= \sum_{z_j} q_j \left[\sum_{z_{i\neq j}} \log \mathrm{P(z,x)} \prod_{i\neq j} q_i\right] - \sum_z q_j \prod_{i\neq j} q_i \left[\log q_j + \sum_{z_{i\neq j}} \log q_i\right]$$

$$= \sum_{z_j} q_j \left[\sum_{z_{i\neq j}} \log \mathrm{P(z,x)} \prod_{i\neq j} q_i\right] - \sum_{z_j} q_j \log q_j \underbrace{\sum_{z_{i\neq j}} \prod_{i\neq j} q_i}_{\Sigma_{\mathrm{x}} P(x)=1} - \underbrace{\sum_{z_j} q_j}_{\Sigma_{\mathrm{x}} P(x)=1} \underbrace{\sum_{z_{i\neq j}} \prod_{i\neq j} q_i \sum_{z_{i\neq j}} \log q_i}_{\mathrm{const}}$$

$$= \sum_{z_j} q_j \left[\sum_{z_{i\neq j}} \log \mathrm{P(z,x)} \prod_{i\neq j} q_i\right] - \sum_{z_j} q_j \log q_j - const$$

Die Gleichung kann als KL-Abweichung zwischen $P^{\sim}(x,z) = \mathbb{E}_{i\neq j}[\log P(z,x)]$ und q_j interpretiert werden. Die Maximierung des ELBO ist gleichzusetzen mit der Minimierung der negativen KL-Abweichung und es ergibt sich Folgendes:

$$\log q_j^*(z_j) = \mathbb{E}_{i\neq j}[\log P(z,x)] + const$$

Jeder Term, der unabhängig von der Zufallsvariable z_j ist, gilt als Konstante und ist unabhängig von der Optimierung. Bei einem Optimierungsschritt wird eine Distribution angepasst und die anderen werden fixiert bzw. eingefroren. Das ist vergleichbar mit dem EM-Algorithmus, allerdings werden die Distributionen einzeln und nicht zusammen optimiert. Dadurch ist prinzipiell die VI-Methode schneller als der EM-Algorithmus. Dies erlaubt es, VI Methoden auch auf große

[40] Vgl. Bishop (2006), S. 464.
[41] Vgl. Bishop (2006), S. 464 ff.

Datenbestände anzuwenden. Ein Anwendungsgebiet ist beispielsweise Topic Modeling, bei dem Dokumente, wie z.B. Nachrichtenfeeds, analysiert und geclustert werden.

Ein Nachteil ist, dass VI-Methoden über die KL-Abweichung optimiert werden. Ein Charakteristikum dieser Methode ist, dass sie asymmetrisch ist und wir verschiedene Ergebnisse erhalten, wenn wir die Terme vertauschen. Eine weitere charakteristische Eigenart ist, dass die KL-Abweichung, z.B. von q und p, versucht, die Wahrscheinlichkeitsmasse von q dorthin zu verlagern, wo die Wahrscheinlichkeit von p hoch ist. Dieses Verhalten kann auch anhand der Formel inspiziert werden.

$$D_{KL}(q,p) = \sum_x q(x) \log \frac{q(x)}{p(x)}$$

Die vier Fälle, in denen jeweils q bzw. p groß bzw. klein ist:

q	p	$D_{KL}(q,p)$
Klein	Klein	Klein
Klein	Groß	Klein
Groß	Klein	Groß
Groß	Groß	Klein

Tabelle 1: Verhalten der KL-Abweichung
Quelle: Eigene Daten

Grafisch kann dieses Verhalten folgendermaßen interpretiert werden. Die Approximation q (roter Kreis) versucht Bereiche zu vermeiden in denen p klein ist. Auf der rechten Seite ist der Sachverhalt umgekehrt. Beide Szenarien sind nur bedingt optimal, da Bereiche ein- bzw. ausgegrenzt werden. [42]

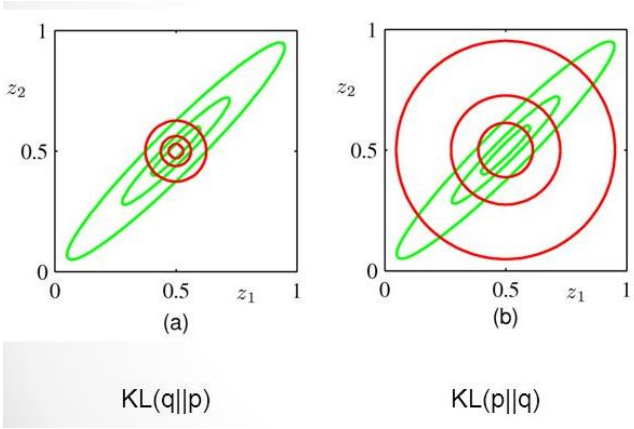


Abbildung 11: KL-Abweichung und negative KL-Abweichung
Quelle: in Anlehnung an Bishop (2006), S. 468.

Eine Folge des „Mean field"-Ansatzes ist, dass keine Korrelation zwischen den Zufallsvariablen abgebildet werden können. Die Abbildung auf der rechten Seite zeigt das Ergebnis einer negativen KL-Abweichung bzw. wenn die Terme vertauscht sind.

[42] Vgl. Bishop (2006), S. 468.

Ein weiterer Nachteil ist, dass die Initialisierung entscheidend ist und die VI Methode nur einzelne Modale erfassen kann. Je nachdem welches Modal erfasst wurde, unterscheiden sich die Resultate. Der Algorithmus konvergiert, allerdings zu einem lokalen Optimum.[43]

Es sollte noch erwähnt werden, dass es andere Methoden gibt, um das Inferenzproblem zu lösen. Beispielsweise bietet die Markov-Chain-Monte-Carlo-Methode (MCMC) eine Alternative. MCMC-Methoden sind exakter, aber sind auch langsamer im Vergleich mit VI-Methoden. Die VI-Methoden lassen sich also einfacher auf große Datenbestände anwenden.[44]

[43] Vgl. Bishop (2006), S. 468 f.

[44] Vgl. Salimans/ Kingma/ Welling (2014), S. 1.

4 Statistisches maschinelles Lernen

In den vergangenen Kapiteln haben wir uns mit den Grundzügen des maschinellen Lernens vertraut gemacht und die wesentlichen Grundkenntnisse kennengelernt, die nötig sind, um eine Inferenz basierend auf dem EM-Algorithmus durchzuführen. In diesem Kapitel werden wir diese beiden Themen miteinander kombinieren, um eine spezielle Sorte von neuronalen Netzen zu erstellen, die das Inferenzproblem mithilfe des maschinellen Lernens löst. Die Kombination von statistischen Elementen und Elementen des maschinellen Lernens, umfasst anspruchsvollere Modelle des maschinellen Lernens die für komplexere unüberwachte Problemstellungen eingesetzt werden können.

Wir werden uns mit dem Konzept der Mannigfaltigkeit und der Hauptachsentransformation vertraut machen sowie auf sogenannte Autoencoder generell eingehen und auf eine spezielle Variante davon, die sogenannten Variational Autoencoder die den Kern dieser Arbeit darstellen.

4.1 **Mannigfaltigkeit**

Ein wesentliches Konzept im Bereich maschinelles Lernen und Mustererkennung ist das der Mannigfaltigkeit. Unter einer Mannigfaltigkeit wird eine zusammenhängende Region an Datenpunkten verstanden. Die Formen dieser Regionen sind oftmals unbekannt. stellen aber eine zentrale Rolle im maschinellen Lernen dar, da sie die wesentlichen Informationen über das Objekt enthalten.[45]

Beispielsweise wenn wir ein Spracherkennungssystem erstellt hätten, dass auf ein Signalwort entsprechend reagiert. Dieses Signalwort richtig zu erfassen und auszuwerten erweist sich als äußerst komplexe Aufgabe. Das System darf nicht von der Stimme, Tonlage, Lautstärke, dem Geschlecht, der Aussprache, Hintergrundgeräuschen oder Ähnlichem in der Form beeinflusst werden, dass es nicht mehr funktionsfähig ist. Das System ist gezwungen, sich auf die wesentlichen Merkmale des Wortes zu konzentrieren. Das geschieht dadurch, dass das System die Informationen filtert und irrelevante Informationen verwirft. Das Konzept der Mannigfaltigkeit ist insofern wichtig, da es den Rahmen angibt, in-wie-weit das ungefilterte Wort auf das gefilterte Wort zeigt. Diese Relation macht beispielsweise andere Aufgaben komplex wie das Generieren von Audio oder Bildern, da beispielsweise das zu generierende Wort auf unzählig viele verschiedene valide Formen des Wortes zeigt, die aus unterschiedlichen Sequenzen der Sprache bestehen. Die Mannigfaltigkeit zu lernen stellt also eine wesentliche Aufgabe im Bereich des maschinellen Lernens dar, da sie das Konzept des Generalisierens am ehesten widerspiegelt.

Ein konvolutionales Netz, dass beispielsweise handgeschriebene Zahlen in einem Bild erkennen soll, transformiert das Bild in eine niedrigere Dimension und soll sämtliche Variationen des Bildes, das die gleiche Zahl zeigt, nahe beieinander clustern. Prinzipiell wird der Algorithmus eine Struktur bei den Zahlen finden und ähnliche Zahlen eher beieinander anordnen als Zahlen, die absolut unähnlich sind. Beispielsweise ist der Übergang von der Zahl Eins zu der Zahl Sieben nahezu fließend. Es wäre also zu erwarten, dass diese Cluster eher beieinander liegen als der Verbund von den Clustern Fünf, Sechs, Acht, Neun und Null. Die Zahlen Zwei und Drei erhalten wahrscheinlich eine eigene Region, da sie zu stark von den anderen abweichen. Wobei die Zahl 3 vermutlich näher an der Acht liegt als die Zahl 2. Die Regionen müssen also nicht zwingend ineinander übergehen, jedoch müssen sie ineinander übergehen, wenn sie dasselbe Objekt abbilden. Also anstatt des Wertes der Zahl zu erfassen, könnte ein Algorithmus erfassen, ob es sich auf dem Bild um eine Zahl oder ein nicht zuzuordnendes Rauschen handelt. Also das Bild auf einen eindimensionalen Wert zu reduzieren, der die Wahrscheinlichkeit angibt ob es sich um

[45] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 161 ff.

eine Zahl handelt oder nicht. Das gewünschte Resultat wäre eine zusammenhängende Region, die angibt, ob das Bild eine valide Zahl darstellt. Die folgende Abbildung zeigt eine Kollektion an Vektorrepräsentationen von Bildern des Projekts, auf das wir später weiter eingehen. Es zeigt sehr deutlich den keilförmigen Verlauf und die ersten Cluster-Versuche in der Lernphase.
Die Datenpunkte richtig anzuordnen, also dass die Nachbarn eines Punktes eine ähnliche semantische Bedeutung haben, und die Entfernungen richtig zu bestimmen, wird innerhalb des Lernprozesses eine zentrale Rolle einnehmen.

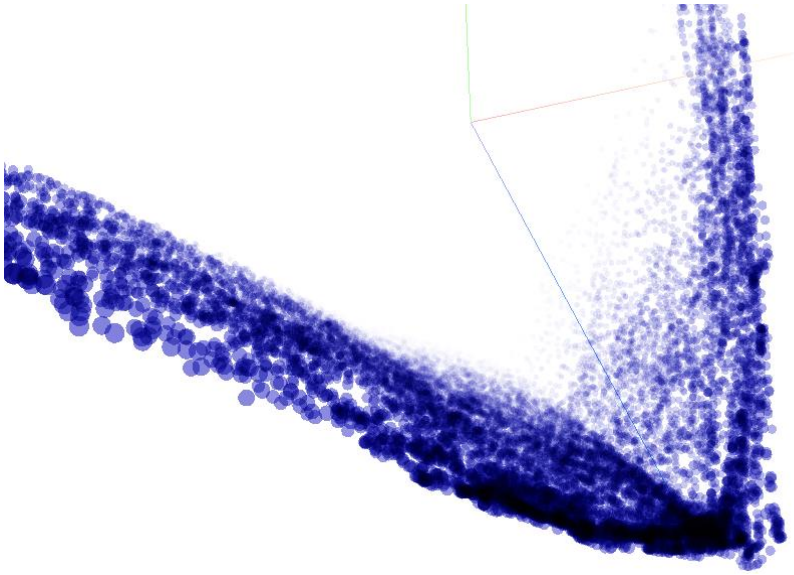


Abbildung 12: Visualisierung der Mannigfaltigkeit
Quelle: Eigene Darstellung

4.2 **Hauptachsentransformation**

Das Konzept der Mannigfaltigkeit machen sich viele Algorithmen des unüberwachten Lernens zu eigen. Ein effizienter und leistungsstarker Algorithmus, der sich durch seine Einfachheit und seiner vielfältigen Einsatzmöglichkeiten auszeichnet ist die Hauptachsentransformation. Formal ausgedrückt sucht der Algorithmus eine lineare Transformation der Daten in eine niedrigere Dimension transformiert mit dem Ziel die Varianz der projizierten Daten zu maximieren.[46] Dabei unterstellt der Algorithmus den Daten, dass eine einfachere Repräsentation in einem Raum in einer niedrigeren Dimension existiert. Beispielsweise könnte die Abbildung in dem vergangenen Abschnitt auch als Ebene interpretiert werden. Sofern wir einen Algorithmus haben, der überprüft, ob wir den kritischen Punkt des Keils erreicht haben, kennen wir die Vorschrift, um die

[46] Vgl. Bishop (2006), S. 561 ff.

Datenpunkte nahezu richtig anzuordnen. Wir haben dadurch die Möglichkeit, dreidimensionale Daten als zweidimensionale Daten abzubilden ohne wichtige Informationen zu verlieren.
Einsatzmöglichkeiten sind beispielsweise die Datenvisualisierung, den Rechenaufwand zu reduzieren indem einfachere Modelle erstellt werden können, Speicherplatz zu reduzieren oder auch um redundante bzw. korrelierende Informationen zu entfernen um dem Algorithmus ein stabileres Training zu ermöglichen.
Die Schritte um die Hauptachsentransformation durchzuführen sind nach Raschka die folgenden:[47]

1. Standardisierung der Daten $x \in \mathbb{R}^d$ für die Menge $\mathbb{X} = \{x_1, \dots, x_n\}$
2. Erstellung der Kovarianzmatrix $\Sigma \in \mathbb{R}^{d \times d}$
3. Zerlegung der Kovarianzmatrix in Eigenvektoren und Eigenwerte
4. Auswahl von k Eigenvektoren, um die größte Varianz in den Daten abzubilden
5. Konstruieren der Projektionsmatrix $W \in \mathbb{R}^{d \times k}$
6. Transformation der Eingabemenge $\mathbb{X}$ mit der Projektionsmatrix W in den Unterraum mit der Dimension k

Der erste Schritt ist die Standardisierung der Daten. Dabei werden die Daten skaliert und oftmals auf einem bestimmten Intervall abgebildet. Dadurch werden Eigenschaften wie beispielsweise die Varianz aussagekräftiger und können mit den Varianzen von anderen Attributen besser verglichen werden.
Der nächste Schritt ist die Erstellung der Kovarianzmatrix. Die Kovarianz gibt Aufschluss über die Relation der Attribute untereinander. Beispielsweise wenn wir die Aktivität eines Computers überwachen würden, könnten wir eine Relation zwischen den Attributen CPU-Auslastung und dem Schreiben bzw. dem Lesen einer Festplatte entdecken. Die Relation ist nicht eindeutig, da das eine nicht zwingend das andere impliziert. Allerdings würde man statistisch diese Relation erfassen können.
Die Kovarianzmatrix Σ ist eine $d \times d$ Matrix in der die einzelnen Elemente wie folgt berechnet werden:

$$\Sigma = \begin{bmatrix} \sigma_{11}^2 & \cdots & \sigma_{1d}^2 \\ \vdots & \ddots & \vdots \\ \sigma_{d1}^2 & \cdots & \sigma_{dd}^2 \end{bmatrix} \; mit \; \sigma_{jk}^2 = \frac{1}{n} \sum_{i=1}^{n} (x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k)$$

Die Elemente der Diagonalen beschreiben die Varianz des jeweiligen Attributs. Außerdem ist die Matrix symmetrisch durch das Kommutativgesetz und es gilt: $\sigma_{jk}^2 = \sigma_{kj}^2$.

[47] Vgl. Raschka (2017), S. 137 – 144.

Wenn die Kovarianzmatrix berechnet ist, können die Eigenwerte sowie die Eigenvektoren ermittelt werden. Dabei wird folgende Gleichung nach λ umgestellt:

$$\Sigma v = \lambda v$$

$$\det\left(\begin{bmatrix} \sigma_{11}^2 - \lambda & \cdots & \sigma_{1d}^2 \\ \vdots & \ddots & \vdots \\ \sigma_{d1}^2 & \cdots & \sigma_{dd}^2 - \lambda \end{bmatrix}\right) = 0$$

Die einzelnen $\lambda_1, \dots, \lambda_d$ die die Gleichung erfüllen, sind die Eigenwerte. Werden die Eigenwerte wieder in die Gleichung eingesetzt, lassen sich die einzelnen Eigenvektoren berechnen.

$$\begin{bmatrix} \sigma_{11}^2 - \lambda_z & \cdots & \sigma_{1d}^2 \\ \vdots & \ddots & \vdots \\ \sigma_{d1}^2 & \cdots & \sigma_{dd}^2 - \lambda_z \end{bmatrix} * v_z = 0$$

Anschließend werden die Vektoren ausgewählt, die am ehesten die Daten widerspiegeln und somit die höchste Varianz aufweisen. Die größten Eigenwerte erfüllen diese Eigenschaft. Formal lässt sich sagen, dass die Summe der Eigenwerte die maximale Varianz abbilden und jeder Eigenwert einen Beitrag an Informationen liefert. Dieser Beitrag an Informationen ist der Eigenwert geteilt durch die Summe der Eigenwerte:

$$\frac{\lambda_j}{\sum_{j=1}^{d} \lambda_j}$$

Die Projektionsmatrix $W = [v_1, \dots, v_k]$ wird durch das horizontale Aneinanderreihen der ausgewählten Eigenvektoren erstellt. Mit dieser Projektionsmatrix wird der Datensatz x auf eine neue reduzierte Repräsentation transformiert:

$$x' = Wx$$

x‘ stellt dabei den neuen transformierten bzw. reduzierten Datensatz dar

Bei der Hauptachsentransformation handelt es sich allerdings um eine lineare Transformation. Sie ist also nur bedingt bei nicht linearen Verläufen einsetzbar. Ein weiteres Problem sind statistische Ausreißer. Bei der Standardisierung werden diese nicht weiter berücksichtigt und so können falsche Skalierungen angewendet werden, die das ursprüngliche Bild zu weit verzerren.

Die Idee dahinter, zur Transformation von aussagekräftigen Features, lässt sich auch in einem Ansatz des maschinellen Lernens darstellen in Form von Autoencodern.

4.3 **Autoencoder**

Eine spezielle Variante von neuronalen Netzen sind Autoencoder. Diese können in einer Problemstellung des ungestützten Lernens eingesetzt werden, um wichtige relevante Merkmale in den Daten zu entschlüsseln. Vereinfacht ausgedrückt komprimieren diese Netze die Daten in

eine Repräsentation und rekonstruieren wieder die Eingabe.[48] Autoencoder haben historisch betrachtet einen besonderen Stellenwert und sind maßgeblich an dem heutigen Erfolg von neuronalen Netzen beteiligt. Restricted-Boltzman-Machines sowie Deep Belief Networks waren frühere Forschungsthemen und haben die Entwicklung von neuronalen Netzen in den Vordergrund gerückt. In der Literatur werden Autoencoder in einem probalistisches Setting auch als „latent variable models“ bezeichnet, da sie eine nicht observierbare Repräsentation beherbergen, die es gilt zu entschlüsseln.

Der Autoencoder besteht dabei im Wesentlichen aus zwei Komponenten, einer Kodierungsfunktion und einer Dekodierungsfunktion. Die Kodierungsfunktion ist verantwortlich relevante Informationen zu finden, die nützlich sind, um es dem Dekodierer zu ermöglichen, die ursprüngliche Eingabe wiederherzustellen.

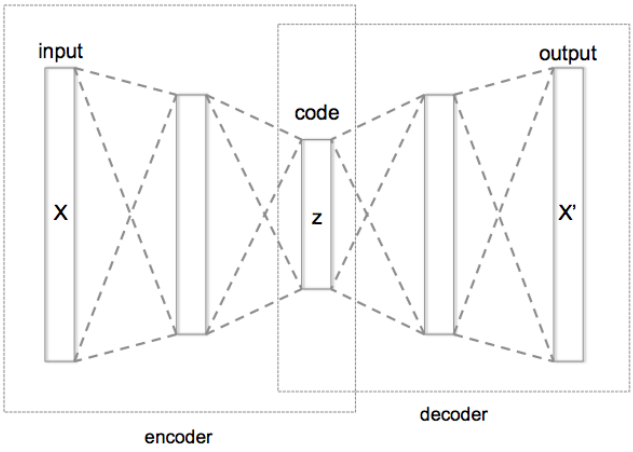


Abbildung 13: Autoencoder Struktur
Quelle: Chervinskii (2015)

$$g(f(x)) = x \; und \; die \; Performancefunktion \; \mathcal{L}\left(x, g(f(x))\right)$$

Dabei ist f eine Kodierungsfunktion und g eine Dekodierungsfunktion.

Der Vektor, der die codierte Form des Inputs enthält, wird in dem Zusammenhang mit Autoencodern auch als embedding bezeichnet. Dadurch, dass es sich um einen Vektor handelt, können sämtliche mathematische Operationen durchgeführt werden, wie die Messung der Distanz zwischen zwei embeddings durch die Berechnung der Länge des Differenzvektors.[49]

Die Struktur der Autoencoder kann wie auch bei neuronalen Netzen willkürlich gewählt werden. Es können Deep Autoencoder mit mehreren versteckten Schichten erstellt werden, die komplexe, willkürliche, nicht lineare Funktionen mit Aktivierungsfunktionen erlernen können oder

[48] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 502.
[49] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 552.

es können auch oberflächliche lineare Funktionen erlernt werden, deren Ergebnisse vergleichbar sind mit der Hauptachsentransformation. Je nach Aufgabenkomplexität, Anzahl der Datensätze sowie Rechenkapazitäten kann ein geeignetes Modell gewählt werden. Das charakteristische Merkmal eines Autoencoders ist vermutlich die sanduhrförmige Struktur, die in der Mitte die codierte Repräsentation enthält. Diese Mitte wird auch als Flaschenhals (engl. Bottleneck) bezeichnet. Die Wahl der Dimension dieser Repräsentation kann sich in der Praxis als äußerst schwierig erweisen. Beispielsweise wenn die Dimension gleich oder größer der Dimension der Eingabeschicht ist, wird der das neuronale Netz dazu neigen, die Werte einfach zu kopieren und überflüssige Neuronen nicht weiter zu verwenden. Das Ergebnis ist nicht zufriedenstellend, da wir nur eine bzw. zwei Identitätsfunktion erlernen. Autoencoder, deren Dimension der Repräsentation größer oder gleich der Eingabeschicht sind, werden auch als übervollendet (engl. overcomplete) bezeichnet. Ist die Dimension kleiner, werden sie als untervollendet (engl. undercomplete) bezeichnet. Untervollendete Autoencoder haben in der Regel einen bedeutsameren Stellenwert als übervollendete da sie zum lernen von Features verwendet werden können.[50]

Bei dem designen des Autoencoders muss allerdings muss ein Equilibrium zwischen der Repräsentations- und der Machtkapazität des Dekodierers bestehen. Es wäre beispielsweise möglich, sämtliche Eingaben des Kodierers als Index auf einer einzelnen Dimension abzuspeichern. Ein zu mächtiger Dekodierer könnte trotz dieses Fehlverhaltens aus diesem Index die passende Eingabe wiederherstellen. Allerdings lernt der Autoencoder keine vernünftige Struktur und kann somit nur als Hashfunktion verwendet werden, der Autoencoder leidet an einer Überanpassung und würde bei neuen Daten komplett versagen.[51]

In der Praxis haben sich einige Varianten von Autoencodern hervorgetan, die zusätzlich zu dem Lernen einer kodierten Repräsentation einige Einschränkungen (engl. Constraints) besitzen.

Eine dieser Formen sind Sparse-Activation-Autoencoder. Diese spezielle Form sucht eine Repräsentation, die nur spärlich aktiviert ist bzw. wo viele Elemente Null sind. Diese Form folgt dem biologischen Paradigma, dass die Neuronen nicht bei jeder Eingabe ein wenig aktiviert werden, sondern nur bei speziellen Eingabemustern stark angesprochen werden. Das Ziel dahinter ist, dass die einzelnen Neuronen eindeutige Charakteristika entdecken, die sich von den anderen unterscheiden. Es sollen also spezialisierte Neuronen gefunden werden, die für eine bestimmtes Szenario gebraucht werden. [52]

Eine weitere Form von Autoencoder sind rauschfilternde Autoencoder. Dabei wird die Eingabe des neuronalen Netzes mit einem zusätzlichen Noise-Vektor verunreinigt und der Autoencoder

[50] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 503 f.
[51] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 504.
[52] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 227.

muss das Rauschen beseitigen, um den ursprünglichen, nicht verunreinigten Wert wiederherzustellen. Diese Form kann beispielsweise dazu verwendet werden, um fehlerhafte Bilder wiederherzustellen oder auch zum Lernen von captcha-Bildern bzw. die captcha-Verunreinigung zu entfernen. Präziser ausgedrückt lernen die Autoencoder ein Vektorfeld das an jeder Stelle einen geeigneten Vektor finden soll, um die verunreinigte Eingabe auf die ursprüngliche Mannigfaltigkeit abzubilden. [53]

$$\mathcal{L}\left(x, g\left(f(x+\epsilon)\right)\right)$$

4.4 **Variational Autoencoder**

Eine besondere Variante von Autoencodern sind Variational Autoencoder nach Kingma und Welling.[54] Dabei werden die deterministischen Autoencoder in ein probalistisches Setting überführt. Der Kodierer wird eine Wahrscheinlichkeitsverteilung von $P_{Kodierer}(z|x)$ und der Dekodierer wird zu einer Wahrscheinlichkeitsverteilung von $P_{Dekodierer}(x|z)$.

Wir können von X auf Z schließen und umgekehrt. Der Autoencoder soll diese Repräsentation finden, um einen unvollständigen Datensatz X zu komplettieren, $P_{Model}(x, z)$. Die Grundannahme ist, dass es in dem Datensatz X versteckte, latente (fehlende) Variablen gibt, die mit dem X Datensatz korrelieren.

Diese Problemstellung kann als Inferenzproblem aufgefasst werden und mithilfe der variational inference Methode gelöst werden. Diese Form von Autoencodern wird auch als Variational Autoencoder (VAE) bezeichnet.

Variational Autoencoder lernen anstatt einer deterministischen Repräsentation eine bzw. mehrere Distributionen, aus denen eine Repräsentation gezogen wird. Der Kodierer wird zu einem neuronalen Netzwerk das, im Falle einer Normalverteilung, die Werte für den Mittelwert μ und die Varianz σ (oder auch die Kovarianzmatrix Σ) vorhersagt. Die folgende Abbildung zeigt den schematischen Aufbau eines VAE:

[53] Vgl. Goodfellow et al (2016), S. 510 ff.
[54] Vgl. Kingma/Welling (2013), S. 1-4.

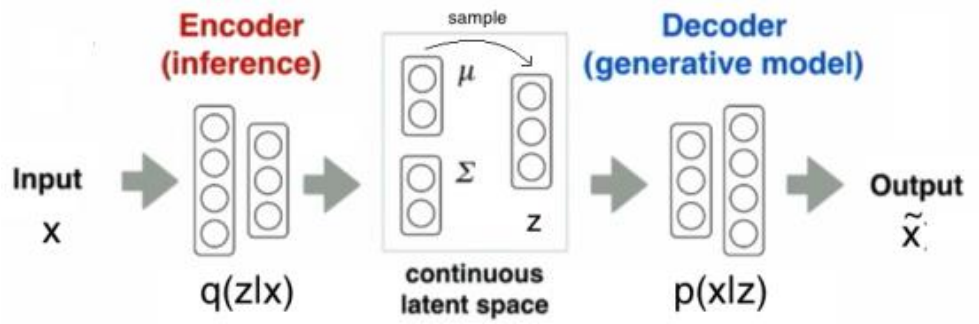


Abbildung 14: Variational Autoencoder
Quelle: Charles H. Martin (2017)

Der Dekodierer zieht aus dieser vorhergesagten Distribution eine Repräsentation und versucht anhand dieser wieder die ursprüngliche Eingabe herzustellen. Wie wir sehen werden, kann diese Vorgehensweise auch als verrauschte Eingabe interpretiert werden und sie gilt als natürlicher Regulator, der das Generalisieren des Autoencoders sicherstellt. Wie in dem Abschnitt über Variational Inferenz beschrieben, verändert sich die Performancefunktion dementsprechend und besteht aus den beiden Termen der ELBO und der KL-Abweichung. Der VAE muss einen Balanceakt lernen zwischen diesen beiden Größen, um das ursprüngliche Ziel des Autoencoders zu erfüllen.

$$ELBO = \mathbb{E}[\log(\mathrm{P(x|z)}] - KL(q(z)||P(z))$$

Die beiden Größen können als Rekonstruktionsfehler und als Abweichung von unserer Modellannahme interpretiert werden. Das neuronale Netzwerk, das die Distribution q vorhersagt, könnte ohne die KL-Abweichung, dazu neigen keine oder nur geringe Varianz vorherzusagen. Dadurch würde das Model eher deterministisch werden und die Zufallskomponente komplett aushebeln. Wenn wir beispielsweise eine Normalverteilung vorhersagen wollen, können wir als Prior $P(z)$ eine Normalverteilung annehmen mit den Werten $\mu = 0$ und $\sigma = 1$. Der VAE wird bestraft, wenn er sich von dieser Verteilung zu weit entfernt. Die Aufgabe ist also, eine Repräsentation auf einem beschränkten Raum zu lernen und die Trainingsbeispiele in diesem Raum anzuordnen. Dadurch, dass Distributionen vorhergesagt werden, entstehen Bereiche, in denen sich die Wahrscheinlichkeitsmasse überlappt. Das ist in dem Sinne ein Problem, da der VAE oftmals nicht unterscheiden kann zwischen zwei verschiedenen Eingaben und somit nicht exakt die Eingabe wiederherstellen kann. Der VAE kann das Problem prinzipiell auf drei verschiedene Arten lösen. Die erste Lösung wäre es, μ so anzupassen, dass sich möglichst wenig Wahrscheinlichkeitsmasse überlappt. Mit anderen Worten: Verschiebe die Distribution nach links bzw. nach rechts und verringere die gemeinsame Wahrscheinlichkeit. Allerdings verstößt dieses Verhalten gegen die weiche Beschränkung der KL-Abweichung. Das Gleiche gilt auch für den zweiten Ansatz der Reduzierung der Varianz, bzw. die Zufallskomponente zu entfernen, um möglichst deterministische Repräsentation zu erhalten, welches ebenfalls gegen die KL-Abweichung

verstößt. Die letzte Möglichkeit ist es, ähnliche Objekte nahe beieinander anzuordnen. Diese Objekte unterscheiden sich beispielsweise in der x-y-Position, der Größe, der Rotation und Ähnlichem. Es muss dazu allerdings noch erwähnt werden, dass die latenten Variablen keine expliziten physikalischen Interpretationen aufweisen müssen.[55] Es werden nur Features entdeckt, die für die Dekodierung am hilfreichsten sind. Der Dekodierer muss diese selbstgewählten Variablen nur richtig interpretieren. Dieser Ansatz verstößt gegen keinerlei Beschränkungen und sorgt dafür, dass (hoffentlich) plausible Repräsentationen gelernt werden. Die nachfolgende Abbildung zeigt die Problemstellung samt den ersten beiden Lösungsansätze anhand von zwei Trainingsbeispielen x1 und x2.

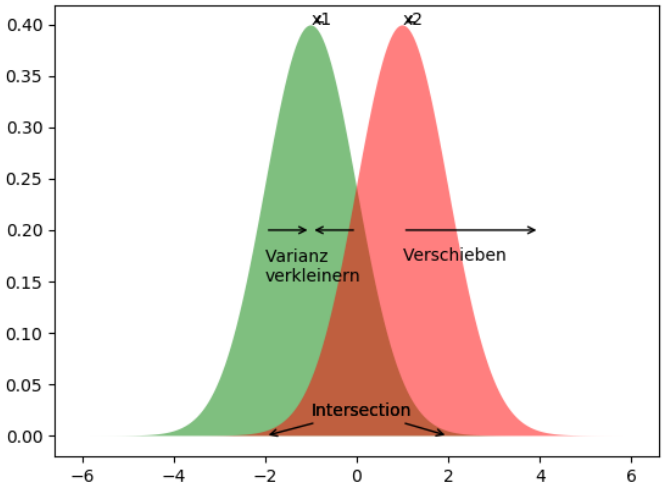


Abbildung 15: Lösungsansätze des Variational Autoencoders
Quelle: Eigene Darstellung

Der VAE gilt als eine verlässlichere Form des Autoencoders, welches ein stabileres Training ermöglicht. Da der VAE eine Distribution lernt, wird er in der Literatur auch als generatives Modell bezeichnet. Die Idee eines generativen Models ist es, vereinfacht ausgedrückt, aus einer Distribution ein Sample ziehen zu können und anhand dieses Samples Inhalte generieren zu können, wie beispielsweise Texte, Bilder, Töne und Ähnlichem.

Für die Generierung von Inhalten wäre es beispielsweise von Vorteil, wenn die Attribute bzw. die Elemente des Samplevektors keinerlei Relation aufweisen. Allerdings ist das in den meisten Fällen nicht möglich. Beispielsweise wenn wir ein generatives Model hätten, das einen Kreis auf einer Leinwand zeichnet, könnten beispielsweise die Farbe des Kreises entscheidend sein, die Größe, die x-Position und die y-Position. Wenn unsere Trainingsdaten allerdings daraus bestehen, dass wir einen Kreis in einer linearen Bewegung vor- und zurück bewegen, könnte das Model genau ein Feature lernen, nämlich den Verlauf dieser linearen Bewegung, da anstelle von

[55] Vgl. Bishop (2006), S. 366.

zwei Attributen nur eins effektiv genutzt werden würde. Das Feature würde die Steigung (bzw. der Vektor bei vertikaler Bewegung) sein und es wäre nicht möglich, den Kreis anders anzuordnen. In der Literatur spricht man in diesem Zusammenhang auch von entangled (verstrickte) und disentangled (entwirrte) Repräsentationen. Die folgende Abbildung zeigt diese Problemstellung an dem Beispiel von handgeschriebenen Zahlen:

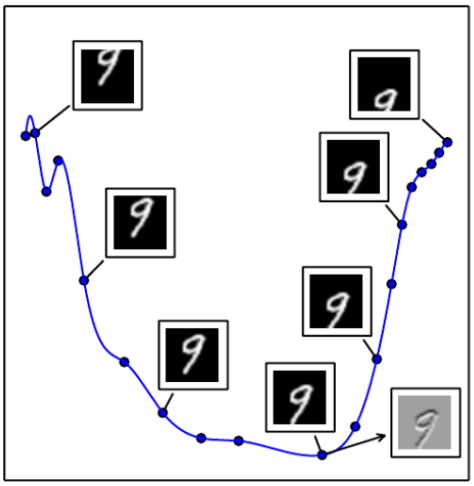


Abbildung 16: Entangled Repräsentationen
Quelle: Goodfellow/ Bengio/ Courville (2016), S. 517.

Obwohl sich nur ein Attribut, die y-Position, ändert beeinflussen zwei Werte die Position. Es wäre beispielsweise hilfreich eine Repräsentation zu finden in der ein einzelnes Attribut für die y-Position verantwortlich ist. Eine Möglichkeit, um dieses Verhalten zu forcieren, sind sogenannte Beta-VAE. Sie funktionieren auf nahezu die gleiche Weise wie normale VAE, allerdings wird die Performancefunktion leicht angepasst und es wird eine stärkere Präferenz für die Beschränkung der KL-Abweichung genommen.[56]

Mit anderen Worten, die KL-Abweichung bekommt mehr Gewicht bei der Performancefunktion als der Rekonstruktionserror.

$$ELBO = \mathbb{E}[\log(\mathrm{P}(\mathrm{x}|\mathrm{z})] - \beta * KL(q(z)||\, P(z))$$

Der Wert für β ist üblicherweise größer oder gleich 1. Das Verhalten lässt sich so interpretieren, dass der Balanceakt zugunsten der KL-Abweichung entschieden wird. Es ist für den Algorithmus erstrebenswerter, größere Bereiche Wahrscheinlichkeitsmasse einzubeziehen und die Varianz ähnlich zu dem Prior zu machen. Gleichzeitig werden weniger Feature benutzt, da es von Nachteil ist, von dem Prior zu weit abzuweichen. Ein Nachteil könnte allerdings sein, dass der β – VAE multiple Distributionen benutzt um ein Feature vorherzusagen, da mit jeder Verteilung ein Stück der Zufallskomponente entfernt wird und aus dem Zufallswerts ein Durchschnitt bzw. ein Erwartungswert wird.

[56] Vgl. Higgins et al. (2016), S. 2 f.

Ein technisches Detail bei der Implementierung von VAE ist die Tatsache, dass sich die Gradienten von Distributionen nicht berechnen lassen. Dadurch, dass eine zufällige Sample gezogen wird, ist es nicht möglich den Gradienten zu berechnen, der nötig ist, um die Parameter mithilfe des Gradientenabstiegsverfahren anzupassen. Eine Lösung für dieses Problem ist der Reparameterization-Trick. Der Reparameterization-Trick bedient sich eines mathematischen Tricks, in dem sich die Distributionen anders darstellen lassen. Beispielsweise könnte eine Normalverteilung folgendermaßen umformuliert werden:

$$\epsilon \sim N(x; \mu, \sigma^2) \; = \; \mu + \sigma * \epsilon \; mit \; \epsilon \sim N(0,1)$$

Diese Form lässt sich einfach herleiten, indem die Normalverteilung ausgeschrieben wird und entsprechend umgestellt wird. Der Unterschied zwischen diesen beiden Formen ist, dass die Zufallskomponente ausgelagert ist und die vorhergesagten Parameter sich so isolieren lassen. Dadurch ist es möglich, die Gradienten zu berechnen, um die Parameter dementsprechend anzupassen. Es ist also möglich, Backpropagation anzuwenden und das Modell zu trainieren trotz der Zufallskomponente. [57]

4.5 **State of the Art für generative Modelle**

VAE haben den Fokus der Wissenschaftler erneut auf Autoencoder gelenkt. Insbesondere gelten diese heute immer noch als „State of the Art“ im Bereich des Lernens einer Repräsentation.[58] Allerdings wurden bessere Modelle für generative Modelle entworfen, die in der Lage sind, neue Inhalte zu generieren, die detaillierter sind. Wie beispielsweise „Generative Adversarial Networks“ (GANs).

Ein GAN besteht aus einem generativen Netzwerk und einem Diskriminator-Netzwerk. Die Idee hinter GANs basiert auf einer Art Täuschungsspiel. Das generative Netzwerk versucht dabei neue Trainingsdaten zu erstellen, wohingegen der Diskriminator versucht diese versuchten Täuschungen aufzudecken und zu unterscheiden, ob es sich bei den aktuellen Daten um Täuschungen handelt oder um Daten aus dem ursprünglichen, originalen, Datenbestand handelt. Das generative Modell wird mit der Zeit immer bessere Täuschungen entwerfen und versuchen den Diskriminator zu täuschen. Allerdings wird dieses Vorhaben den Diskriminator gleichzeitig dazu animieren, immer besser zwischen den beiden Distributionen zu unterscheiden. Im Vergleich zu VAE bieten GANs allerdings bessere Resultate. Beispielsweise VAE, die Bilder generieren, tendieren dazu, eher verwaschen zu wirken. Die Ursachen dafür sind noch unbekannt.[59] Im

[57] Vgl. Kingma/Welling (2013), S. 4 f.
[58] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 699.
[59] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 697.

Gegensatz zu VAE tendieren GANs allerdings zu einem deutlich instabileren Training, da das sogenannte Nash-Equilibrium zwischen den Kräften des Diskriminators und des Generators bestehen muss. Ein zu starker Diskriminator wäre in der Lage, die Täuschungen sofort zu erkennen, und der Generator wäre nicht in der Lage, sich zu verbessern, da die Gradienten zu klein sind. Ein zu schwacher Diskriminator wäre für das Endresultat nicht gut genug. Je stärker der Diskriminator unterscheiden, kann desto besser sollten die Resultate werden. [60]

Es gibt zwar einige Tricks, um das Training zu stabilisieren, die sich empirisch bewährt, haben allerdings ist es generell schwieriger, ein GAN als ein VAE zu trainieren und es braucht deutlich mehr Zeit, um brauchbare Resultate zu erzielen. Die Resultate von GANs sind allerdings deutlich besser. Die folgende Abbildung zeigt die verschiedenen Resultate zwischen MSE und einem GAN:

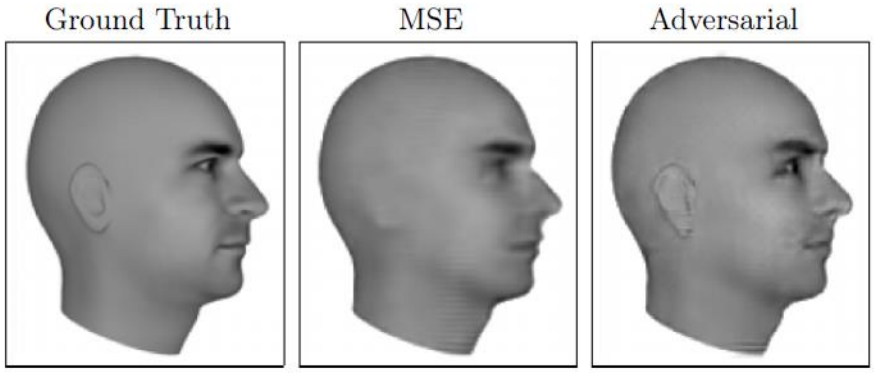


Abbildung 17: Vergleich der Resultate zwischen MSE und GAN
Quelle: Goodfellow/ Bengio/ Courville (2016), S. 545.

Trotzdem gelten VAE immer noch als mächtige Netzwerke, die für eine Vielzahl von Problemstellungen eingesetzt werden können, insbesondere zum Lernen der Mannigfaltigkeit.[61] Eine dieser Problemstellungen ist beispielsweise eine Domain Adaption bzw. dem mappen von einer Domain auf die andere. Die Idee hinter einer Domain Adaption ist, zwei VAE auf zwei unterschiedliche Bereiche zu trainieren. Beispielsweise könnte ein VAE die Mannigfaltigkeit von einem Bild-Objekt lernen, wohingegen ein weiterer VAE antrainiert werden kann, der die Mannigfaltigkeit von Textdaten lernt. Werden diese beiden Netzwerke kombiniert, können Netzwerke erstellt werden, die beispielsweise von einem Text auf ein Bild zeigen und umgekehrt. Es muss allerdings noch ein zusätzliches neuronales Netzwerk zwischengeschaltet werden, das die eine Mannigfaltigkeit auf die andere abbildet, da die beiden Strukturen nicht zwingend identisch sind.

[60] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 699 ff.
[61] Vgl. Goodfellow/ Bengio/ Courville (2016), S. 699.

5 Praxisprojekt: Optimierung von Videos mithilfe von Variational Autoencodern

Diese Arbeit wird sich mit Optimierung von Videos mithilfe von variationalen Autoencodern befassen. Für diese Arbeit wurde ein kurzes, simples Video aufgenommen, in denen einige Operationen einer Hand durchgeführt wurden. Wir werden uns zunächst mit der Datenvorverarbeitung, also mit dem Überführen des Videos in einen geeigneten Datensatz, befassen und anschließend auf das VAE-Model eingehen, das in dieser Arbeit verwendet wurde. Anschließend werden wir uns mit den Ergebnissen sowie der Evaluation befassen, um das Projekt abzuschließen.
In dem Experiment werden wir versuchen, die Bildwiederholrate (engl. Framerate) künstlich mit VAEs zu erhöhen. Dabei werden wir unserem Datenbestand, auf dem wir im folgenden Kapitel eingehen, sukzessiv verringern und überprüfen, ob wir mit einem kleineren Datenbestand, also einer geringen Bildwiederholrate, die Bildwiederholungsrate des Videos künstlich erhöhen können, um flüssigere Videos zu erhalten. Wir werden dabei auf die Problematiken bei dieser Methode eingehen.

5.1 Der Datensatz sowie die Vorverarbeitung

Der Datensatz, der in dieser Arbeit verwendet wird, besteht aus einem ca. 11-minütigem Video, in dem einige Basisoperationen einer Hand gezeigt werden. Das Video wurde dabei vor einem blauen Hintergrund aufgenommen der anschließend mithilfe einer Maske entfernt wurde. Insgesamt besteht der Datensatz aus 19860 Einzelbildern, die auf die Abmaßen von 64x64 Pixeln herunterskaliert wurden. Die Bilder wurden anschließend in eine Graustufe umgewandelt, um das Lernen für den Algorithmus zu vereinfachen, da anstelle der drei Komponenten RGB nur eine kombinierte Komponente verwendet wird, die den Mittelwert der drei Komponenten darstellt. Die folgende Abbildung zeigt den Prozess des Maskierens:

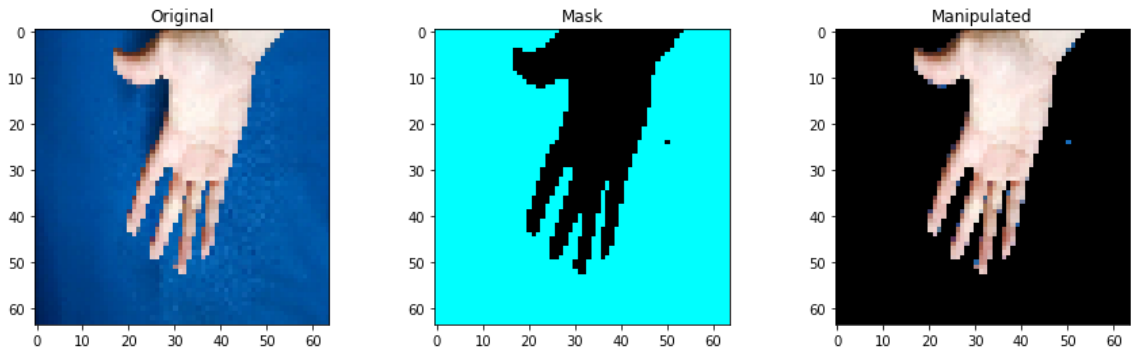


Abbildung 18: Maskieren des Videos
Quelle: Eigene Darstellung

Das Video besteht dabei aus einer zusammenhängenden Aufnahme, in der die Umformung von einem Zustand zu einem anderen eine fließende Bewegung darstellten.

Einige dieser Zustände sind beispielsweise:

- Die Bewegungen der Hand in einem dreidimensionalen Koordinatensystem
- Das Spreizen der Finger,
- Das Bilden einer Faust,
- Das Rotieren der Hand,
- Die ausschließliche Bewegung des Daumens,
- Das Bewegen des ersten Fingerglieds, die anderen Glieder bleiben verborgen.

Da das Video eine fließende Bewegung darstellt, ist es möglich, die Relation zwischen den verschiedenen Zuständen zu erfassen. Beispielsweise schließt das Bilden einer Faust das Spreizen der Finger aus, jedoch nicht die Bewegung im Koordinatensystem. Die folgenden Bilder zeigen einen Auszug aus dem Datenbestand, an denen beispielhaft einige Zustände dargestellt sind.

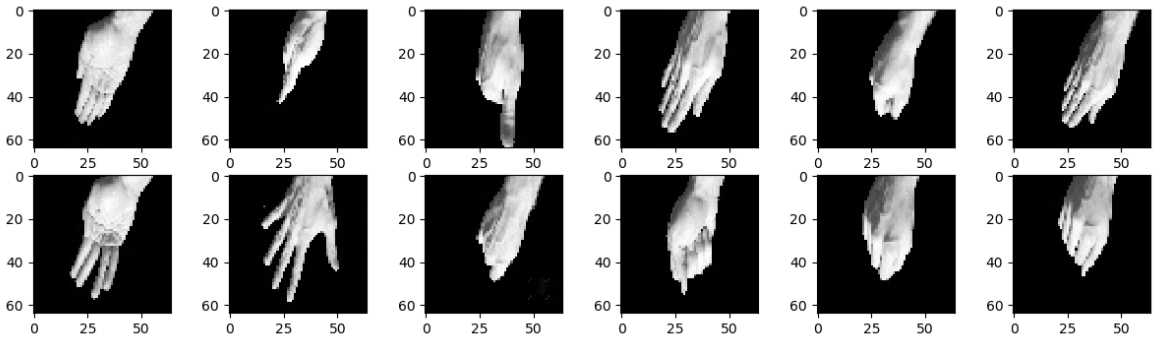


Abbildung 19: Trainingsdaten Beispiele
Quelle: Eigene Darstellung

Wie an der Abbildung erkennbar, sind noch teilweise viele Details sichtbar, wie beispielsweise Knochen, Adern, Lebenslinien, teilweise Schattierungen, Falten usw. Allerdings sollten diese Attribute in Relation zu den einzelnen Zuständen stehen, da beispielsweise bei einer Faust die Knöchel deutlich sichtbarer sind als bei einer ausgestreckten Handfläche. Es ist also möglich

aufgrund der Stellung der Hand der Rotation usw. erahnen zu können wo, und wie stark, der Knöchel sichtbar ist. Dasselbe Prinzip ist selbstverständlich auch auf andere Details anwendbar.

5.2 **Modell**

Der VAE, der in dieser Arbeit verwendet wird, besteht dabei im Wesentlichen aus einem konvolutionalen Netzwerk mit folgender Architektur:

Schicht:	Enthält:
1	1. Konvolutionale Schicht mit 32 Filtern je 5×5×1px und einer Schrittweite von 1 2. ReLU Aktivierungsfunktion 3. Averagepoolingoperation mit einer Filtergröße von 2×2px und einer Schrittweite von 2
2	1. Konvolutionale Schicht mit 64 Filtern je 5×5×32px und einer Schrittweite von 1 2. ReLU Aktivierungsfunktion 3. Averagepoolingoperation mit einer Filtergröße von 2×2px und einer Schrittweite von 2
3	1. Konvolutionale Schicht mit 128 Filtern je 5×5×64px und einer Schrittweite von 1 2. ReLU Aktivierungsfunktion 3. Averagepoolingoperation mit einer Filtergröße von 2×2px und einer Schrittweite von 2
4	1. Neuronale Schicht mit 512 Neuronen 2. ReLU Aktivierungsfunktion
5	1. Neuronale Schicht mit 25 Neuronen. Anhand dieser Schicht werden die Werte für μ und σ vorhergesagt. Unsere Repräsentation für die Eingabe x mithilfe von 25 Werten abgebildet. 2. ReLU Aktivierungsfunktion
6.1	1. Vorhersage von μ mit einer neuronalen Schicht mit 25 Neuronen 2. Als Aktivierungsfunktion wurde die Identitätsfunktion gewählt
6.2	1. Vorhersage von σ mit einer neuronalen Schicht mit 25 Neuronen 2. Softplus Aktivierungsfunktion
7	1. Die Repräsentationsschicht mit einem Sample aus der vorhergesagten Distribution von den Schichten 6.1 und 6.2
8	1. Neuronale Schicht mit 16384 Neuronen 2. Umformung der neuronalen Schicht zu einem 4×4×1024 Tensor 3. Batchnormalization[62] 4. ReLU Aktivierungsfunktion
9	1. Transpose konvolutionale Schicht[63] mit 5×5×512px stride 2

[62] Batchnormalization bezeichnet eine Regulierungsmethode, bei der die Aktivierungen der Neuronen einer statistischen Distribution unterstellt werden. Einzelheiten können unter Ioffe, S./ Szegedy, C. (2015) nachgelesen werden.

[63] Eine transpose konvolutionale Operation ist eine umgekehrte konvolutionale Operation. Anstelle einer Komprimierung der Informationen werden diese dekomprimiert. Einen guten Überblick über diese Operation liefert Dumoulin, V./ Visin, F. (2016), S. 18-23.

	2. Batchnormalization 3. ReLU Aktivierungsfunktion
10	1. Transpose konvolutionale Schicht mit 5×5×256px stride 2 2. Batchnormalization 3. ReLU Aktivierungsfunktion
11	1. Transpose konvolutionale Schicht mit 5×5×128px stride 2 2. Batchnormalization 3. ReLU Aktivierungsfunktion
12	1. Transpose konvolutionale Schicht mit 5×5×1px stride 2 2. Sigmoid Aktivierungsfunktion

Tabelle 2: Aufbau des Modells

Quelle: Eigene Daten

Das Modell besteht aus insgesamt 12 Schichten, bei dem jeweils 5 Schichten für den Kodierer und Dekodierer reserviert sind. Der Flaschenhals besteht aus insgesamt 25 Werten, die das aktuelle Bild repräsentieren sollen. Das oben beschriebene Modell stellt dabei das Modell dar, dass für die Praxis eingesetzt werden kann. Bei der Trainingsphase wird noch eine Schicht bzw. zwei Schichten benötigt, um den Fehlerwert zu bestimmen bestehend aus dem ELBO und der KL-Abweichung des Priors.

5.3 **Durchführung**

Dieses Kapitel soll einen Überblick über die Resultate geben, die hilfreich und notwendig sind, um die späteren Ergebnisse auswerten zu können. Wir werden uns dabei bei der Durchführung auf einen vollständigen Datenbestand beschränken. Wir gehen zunächst auf die gelernte Mannigfaltigkeit des Videos ein und anschließend auf die einzelnen Komponenten der Repräsentation. Wir werden zusätzlich Vorüberlegungen über das Verhalten der Modelle treffen.

5.3.1 Die Mannigfaltigkeit der Hand

Sämtliche Bilder des Videos wurden mit einem separaten Modell als 8-dimensionale Datenpunkte abgespeichert nach ungefähr 100 Trainingsepochen. Auf diesen Datenbestand wurde eine Hauptachsentransformation durchgeführt, um diese 8-Dimensionen auf 3 Dimensionen abzubilden. Diese 3 Dimensionen beschreiben 75,7% der Varianz der Daten. Das bedeutet, dass die 3 Komponenten 75,7% der Informationen der Daten repräsentieren. Wie in den nachfolgenden Abbildungen zu erkennen ist, haben sich eine konkave und eine konvexe Kuppel gebildet, deren Spitzen sich zu überlappen scheinen. Diese zwei Kuppeln umfassen einerseits das Spektrum der Faust samt verschiedener Rotationen inklusive der geschlossenen Handfläche mit gespreizten Daumen und die andere Kuppel beschreibt andere Operationen, die mit der Handfläche durchgeführt werden können, wie das Spreizen der einzelnen Finger sowie die Überführung

der ausgestreckten Handfläche zu einer Faust. Das Zentrum des Gebildes besteht aus Bildern des Handrückens mit nicht gespreizten Fingern. Es ist zu erkennen, dass die Öffnungen der beiden Kuppeln in einem Winkel von ca. 90° versetzt sind. Die X-Y-Z-Dimensionen lassen sich wie folgt interpretieren: Die X-Dimension ist ein Indikator dafür, wie breit das Bild der Hand ist. Befindet sich beispielsweise die Handfläche auf dem Tisch, entspricht dies einer breiten Hand, da diese mehr Raum im Bild einnimmt als eine Hand, deren Unterkante auf dem Tisch liegt. Die Y-Dimension ist schwieriger zu bestimmen. Sie dient meiner Meinung nach dazu die zwei Archetypen der Hand zu trennen. Die Handfläche sowie die Faust. Es ist zu erkennen, dass auf der Unterseite des Gebildes sich eher Handflächen befinden und auf der Oberseite eher Fäuste. Die Z-Dimension scheint auf der Unterseite ein Indikator für das Spreizen der Finger zu sein. Je gespreizter die Finger, desto höher ist dieser Wert. Die Oberseite besitzt allerdings keine auffällige Interpretation, sie besteht aus einer Mischung von Bildern, die eine Faust mit jeweils Vor- und Unterseite zeigen sowie die Oberkante der Hand. Diese Dimension scheint die schwächste zu sein, da deren Ergebnisse weniger Relevanz besitzen. Bei der Hauptachsentransformation entsteht bei der Reduzierung der Daten zwangsläufig ein Datenverlust. Dieser schlägt sich anscheinend überwiegend auf der Z-Dimension aus.

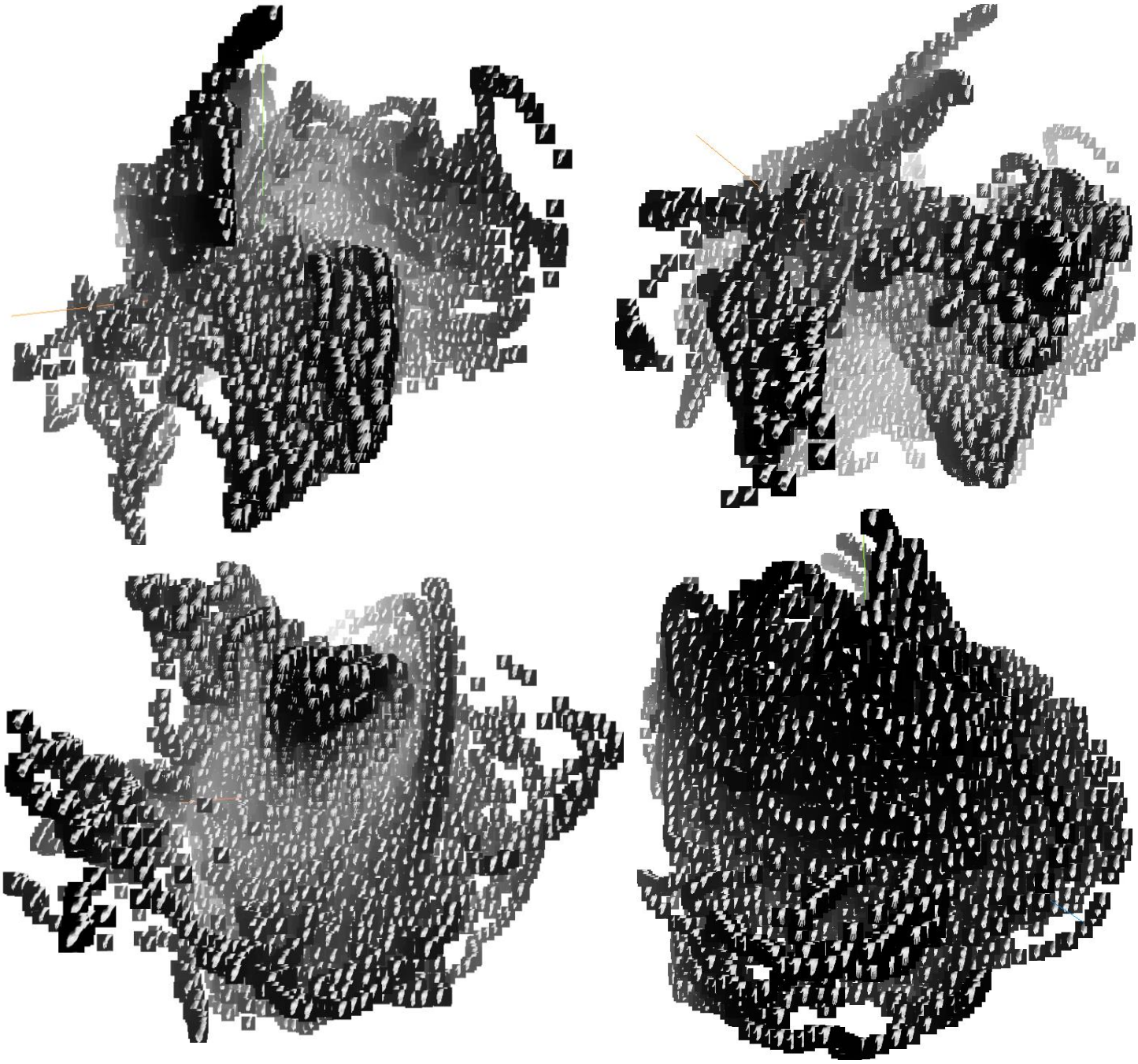


Abbildung 20: Die Mannigfaltigkeit der Hand
Quelle: Eigene Darstellung

Die Abbildung oben links zeigt eine leicht versetzte Frontalansicht der Mannigfaltigkeit, die Abbildung rechts daneben das Seitenprofil. Die unteren Abbildungen zeigen einerseits die Unterseite des Gebildes und die Rückseite, an denen sich die zwei Kuppeln überlagern.

5.3.2 Die Komponenten der Repräsentation

In dem vergangenen Abschnitt haben wir die Mannigfaltigkeit der Hand gesehen. Allerdings wurde die Repräsentation auf einen dreidimensionalen Raum projiziert und somit wurde die Aussagekraft der einzelnen Komponenten unterschlagen. In diesem Abschnitt wollen wir uns näher mit den einzelnen gelernten Komponenten beschäftigen anhand von ausgewählten Beispielen. Präziser ausgedrückt werden wir die Veränderung des wiederhergestellten Bildes illustrieren, wenn wir die einzelnen Komponenten leicht modifizieren. Es sollte unbedingt noch erwähnt werden, dass die ursprüngliche Annahme, dass die einzelnen Komponenten disjunkt sind, sich nicht immer bewerkstelligen lässt. Die Komponenten sind mehr oder weniger abhängig voneinander und es entstehen viele spärliche Bereiche innerhalb der Mannigfaltigkeit. Mit anderen Worten, das Manipulieren von lediglich einer Komponente ist nicht zwingend aufschlussreich für das Endresultat, da die Änderung einer Komponente häufig das Ändern von weiteren Komponenten impliziert. Außerdem wird die Reihenfolge der Komponenten im Laufe des Trainingsprozesses festgelegt. Es ist also sehr wahrscheinlich, dass die Komponenten bei mehrmaligen Trainingsdurchläufen miteinander vertauscht sind, sofern der Zufallsgenerator bei der Initialisierung der Gewichte nicht näher spezifiziert wird.

Die folgende Abbildung zeigen die Ergebnisse, wenn die Komponenten leicht modifiziert wurden. Die einzelnen Zeilen zeigen die Veränderungen, die entstehen, wenn diese eine Komponente manipuliert wird und die anderen einen fixen Wert annehmen. Der Grad der Manipulation entspricht dabei einem Wertebereich von $[-4, 4]$ der mit insgesamt 15 Bildern veranschaulicht wird. Das entspricht einer Schrittweite von ungefähr 0,5 pro Bild. Die Bilder sind von einer negativen- hin zu einer positiven Veränderung angeordnet.

Es ist zu erkennen, dass viele der Komponenten keine oder nur eine sehr geringe Auswirkung auf das Bild zu haben scheinen. Beobachtungen der latenten Plots[64] von 12 verschiedenen Bildern zeigen, dass insbesondere die Komponenten 1, 11, 13, 14 und 19 aktiv sind. Es lässt also den Schluss zu, dass die Repräsentation weiter komprimiert werden könnte bzw. mehr Dimensionen vorhanden sind als eigentlich benötigt. Geringfügige Veränderungen sind beispielsweise leichte Schattierungen oder geringes Wachsen bzw. Schrumpfen der einzelnen Finger. Dieses Ergebnis deckt sich auch mit unserer vorherigen Observation, dass relativ wenig Dimensionen ausreichend sind, um die Informationen des Bildes akkurat widerzugeben. Eine Vermutung besteht, dass die überflüssigen Dimensionen Archetypen der Hand darstellen, um die Zufallskomponente innerhalb der Performancefunktion auszuhebeln. Es wäre beispielsweise denkbar, dass die Gewichte zwischen der nachfolgenden Schicht und der Repräsentation eine gewisse

[64] Siehe Digitalen Anhang.

Ähnlichkeit aufweisen. Eine weitere Möglichkeit wäre, dass es sich bei diesen Komponenten um Experten-Komponenten handelt, die nur sehr wenig verwendet werden. Diese Komponenten wären dann dafür zuständig Flags innerhalb des Generierungsprozesses zu verwenden.
Die folgende Tabelle zeigt observierte Auffälligkeiten innerhalb der jeweiligen latenten Plots.

	0	1	2	3	4	5	6	7	8	9	10	11
1	x	x	x		x	x	x					x
2												
3												
4							x					
5												
6												
7												
8						x						
9						x						
10												
11	x	x	x			x	x	x	x			x
12												
13	x	x	x		x	x	x	x	x	x		x
14	x	x	x		x	x	x	x	x			x
15												
16			x		x	x	x					x
17												
18												
19	x	x	x	x	x	x	x	x	x	x	x	x
20												
21												
22												
23												
24												
25												

Tabelle 3: Auffälligkeiten der latenten Plots
Quelle: Eigene Daten

Die nachfolgende Abbildung zeigt den latenten Plot mit den meisten Auffälligkeiten nach 90 Trainingsepochen mit einem vollständigen Datenbestand. Jede dieser Trainingsepochen besteht aus ungefähr 310 Minibatch-Einheiten, die zufällig zusammengestellt wurden. Das entspricht ungefähr 28000 Optimierungsschritten.

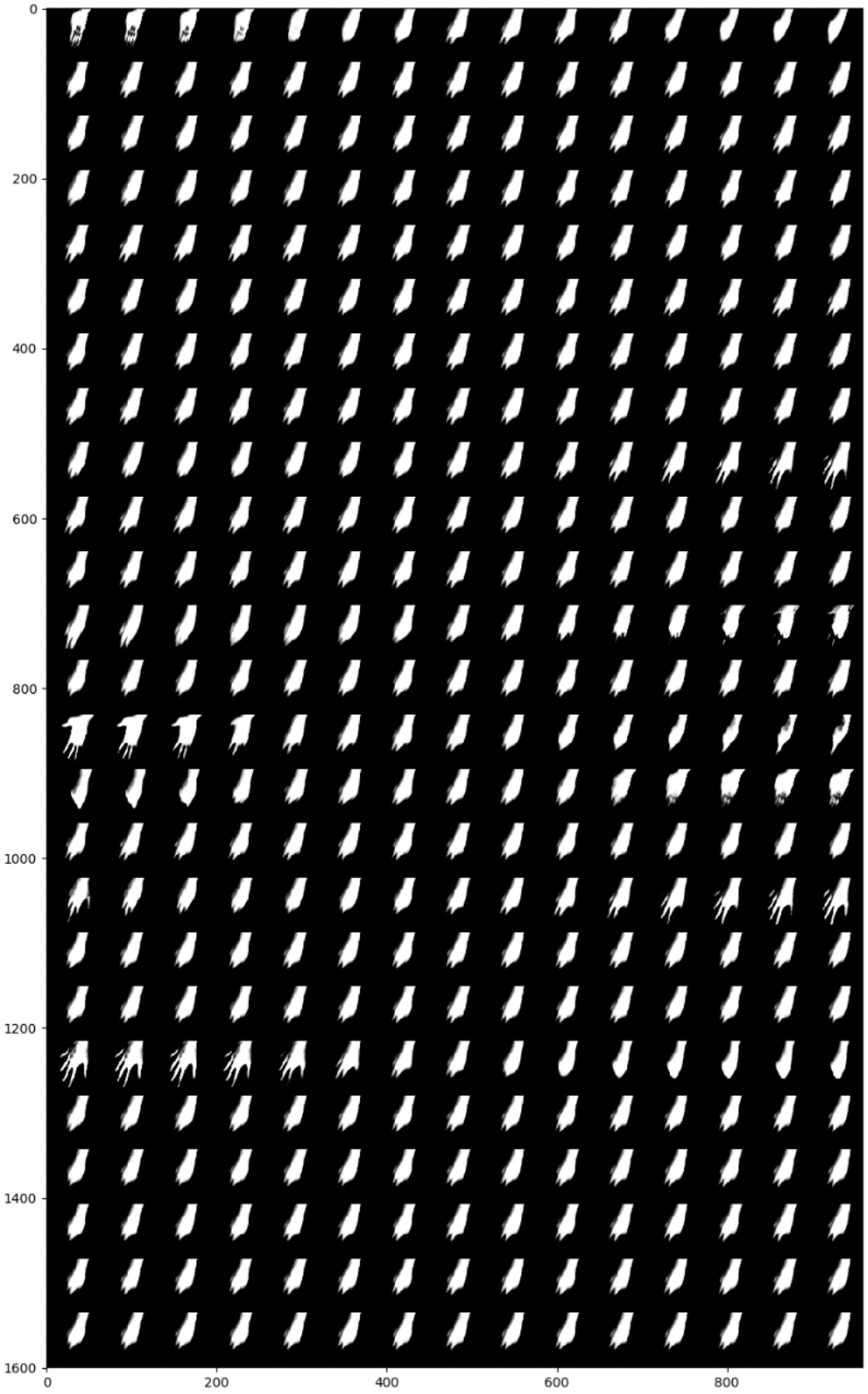


Abbildung 21: Latenter Plot Nr. 5
Quelle: Eigene Darstellung

5.4 **Optimierung von Videos**

Wie anfangs beschrieben, besteht die Aufgabe dieser Arbeit in der Optimierung von Videos mithilfe von VAE. Im vergangenen Abschnitt haben wir uns mit der gelernten Repräsentation beschäftigt, die der VAE lernt um das Bild wiederherzustellen. In diesem Abschnitt gehen wir auf den Versuchsaufbau, die Probleme und die Ergebnisse der Experimente ein. Abschließend geben wir ein Fazit über die Eignungsfähigkeit dieser Methode.

5.4.1 Versuchsaufbau

Das Experiment sieht vor, den vollständigen Datenbestand auszudünnen und nur einen Bruchteil des Datenbestandes im Trainingsprozess zu verwenden. Das Video dieses ausgedünnten Datenbestandes wieder vollständig zu rekonstruieren stellt das zentrale Ziel dieses Projekts dar. Mit vorheriger Überlegung zu der Mannigfaltigkeit sollte es möglich sein, zwischen den einzelnen Datenpunkten zu interpolieren und neue, bis dato unbekannte Bilder zu generieren. Wir werden insgesamt neun verschiedene VAE-Models mit einem reduzierten Datenbestand trainieren. Das Verhältnis zwischen verworfenen und dem verwahrten Bild lässt sich anhand der Nummer bestimmen. Die einzelnen Bilder wurden vor dem Mischen des geordneten Datenbestandes entfernt, um eines nicht flüssiges Video zu simulieren. Das ursprüngliche Video besteht aus 30 Bildern pro Sekunde. Es werden ungefähr 6000 Optimierungsschritte pro Modell angesetzt, um die Ergebnisse besser miteinander vergleichen zu können. Die spezifischen Merkmale der einzelnen Trainingseinheiten lassen sich aus der folgenden Tabelle entnehmen:

Model	Datenbestand	Frames/s	Minibatches	Trainingsepochen
2	9930	15	155	40
3	6620	10	103	60
4	4965	7,5	78	80
5	3972	6	62	100
6	3310	5	52	120
7	2837	4,3	44	140
8	2483	3,8	39	160
9	2207	3,3	34	180
10	1986	3	31	200

Tabelle 4: Datenbestände der einzelnen Modelle
Quelle: Eigene Daten

Die Bezeichnung des Modells ist aufgrund der Übersichtlichkeit um einen Wert verschoben.
Anhand der Tabelle und der zugrundeliegenden Formel lässt sich ablesen, dass die einzelnen Modelle sich nicht linear unterscheiden und die Funktion gegen Null konvergiert. Der Sprung zwischen den einzelnen Modellen nimmt immer weiter ab. Beispielsweise unterscheidet sich die

Ausgangslage von Model 10 und 9 nur geringfügig, wohingegen der Sprung von 2 zu 3 gravierender ausfällt.

Die folgende Abbildung zeigt die Leistungen der einzelnen VAE-Models in dem Trainingszeitraum anhand der Performancefunktion. Die einzelnen Werte wurden jeweils an den Epochenenden gespeichert. Die Modelle einer höheren Nummer besitzen also eine verbesserte Präzision, allerdings lassen sich die stochastischen Verläufe erkennen.

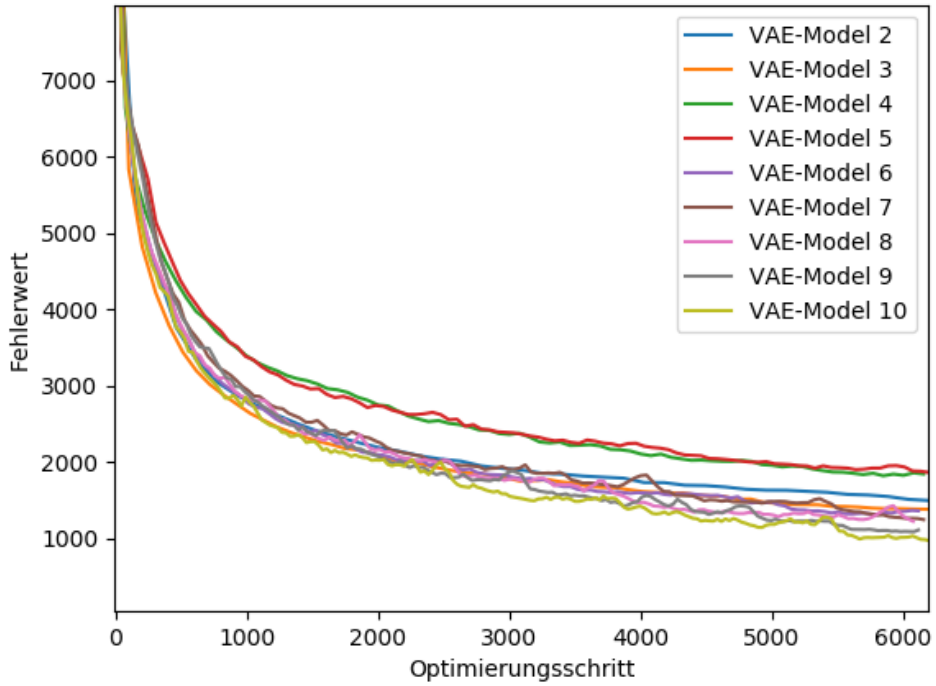


Abbildung 22: Performancemessungen der einzelnen Modelle
Quelle: Eigene Darstellung

Jedes der VAE-Modelle erstellt ein Video, dass sich in vier Areale einteilen lässt. Die obere linke Ecke stellt die „Ground Truth“ dar bzw. das vollständige Video, das der VAE vorhersagen soll. Die untere Zeile entspricht dem reduzierten Video. Auf der linken Seite kann man bei VAE mit einer höheren Nummer sehr gut das Stottern beobachten. Die rechte Seite hingegen bildet den eigentlichen Kern dieser Arbeit. Es ist das vorhergesagte Video, das eine künstliche höhere Bildwiederholungsrate besitzt. Um die Bildwiederholungsrate zu erhöhen wurde zwischen den einzelnen Bildern linear interpoliert. Es wird der Differenzvektor berechnet, der anschließend mit einem Bruchteil multipliziert wird in Abhängigkeit des vorhersagten Frames und anschließend zu dem Ausgangsframe hinzuaddiert wird:

$$Frame_{Neu} = Frame_i + (Frame_{i+1} - Frame_i) * \frac{Interpolationsschritt}{VAE_{Nr}}$$

Bei der linearen Interpolation handelt es sich um eine der einfachsten Varianten der Interpolation. Es existieren zahlreiche weitere Verfahren, die je nach Anwendungsgebiet geeigneter sind.

Beispielsweise entstehen bei der linearen Interpolation knicke an den einzelnen Datenpunkten des Graphen. Es handelt sich also um einen nicht differenzierbaren Graphen. Kompliziertere Verfahren sind allerdings in der Lage, die Interpolation weicher durchzuführen. Allerdings soll die Interpolation nicht das Hauptaugenmerk dieser Arbeit werden.

5.4.2 Beobachtbare Probleme

Wir wollen uns zunächst einen Überblick verschaffen über generelle Probleme bei dieser Methode sowie die Ursachen diskutieren. Wir werden im darauffolgenden Abschnitt die Videos nach genau diesen Kriterien hin untersuchen und erfassen, ob die Probleme amplifiziert werden mit steigender Nummer oder ob die Effekte erst ab einer bestimmten Bildwiederholungsrate observierbar werden.
Die folgende Abbildung zeigt einen beispielhaften Ausschnitt eines Videos:

Abbildung 23: Unschärfe des Videos
Quelle: Eigene Darstellung

Es lässt sich bei den Videos generell eine gewisse Unschärfe des generierten Bildes feststellen. Die Schattierungen sowie die Adern sind nur leicht sichtbar. Dieses Problem war, wie im Abschnitt über VAE beschrieben, zu erwarten. Allerdings sind auch Verbesserungen zu beobachten. Beispielsweise haben die Finger auf der linken Seite einen eher gestuften Verlauf, wohingegen auf der rechten Seite ein Glättungseffekt eintritt. Der Unschärfe-Effekt könnte reduziert werden, indem wir die Performancefunktion leicht anpassen würden. Beispielsweise könnten wir, mit der Annahme dass der Kodierer präzise ist, sämtliche Werte in der Repräsentation abspeichern und einen neuen Trainingsdatenbestand bilden. Dieser Datenbestand kann daraufhin verwendet werden, um den Generator erneut zu trainieren und nur den Rekonstruktionsfehler in

Betracht zu ziehen. Mit anderen Worten, die KL-Abweichung bekommt keine Relevanz mehr und es wird nur noch nach dem Rekonstruktionsfehler optimiert.

Es lassen sich noch zwei weitere deutliche Effekte beobachten. Es entstehen bei einigen Szenen Verunreinigen. Diese Verunreinigungen könnten Artefakte des Maskierungsprozesses sein. Die einzelnen Pixelpunkte sind schwer zu optimieren, da es sich bei der Performancefunktion um eine Formel quadratischer Natur handelt. Je größer die Differenz, desto größer ist der Beitrag an der Performance. Zusätzlich ist zu dem einzelnen Pixel die Koordinate notwendig, um exakt an dieser Stelle eine Pigmentierung durchzuführen. Allerdings treten diese Effekte in größeren Arealen auf und lässt den Schluss zu, dass der VAE schwarze Farbe auf einen weißen Hintergrund aufträgt als umgekehrt. Die Erklärung könnte auch die Bildung von Schattierungen erklären, da die einzelnen Schattierungen überlagert werden können und somit die Intensität des Pixels bzw. des Pixelbereichs reduzieren können. Das Problem könnte vermutlich mit einem professionellem Videosetting reduziert werden, allerdings ist gerade diese Herausforderung für den VAE interessant zu beobachten. Eine weitere Problematik, die häufig bei Modellen mit einer höheren Nummer auftreten, sind teilweise groteske Verformungen der Hand. Dieser Effekt lässt sich häufig bei abrupten Veränderungen der Hand beobachten. Vermutlich ist dieser Effekt eng an die Interpolationsmethode gekoppelt, da wir, respektive der Mannigfaltigkeit, unzulässige Bereiche nutzen. Das ist vergleichbar mit dem Einsetzen eines Wertes, der sich nicht im Definitionsbereich einer Funktion befindet. Das Problem bei der Interpolation ist, dass zwischen den Einzelbildern keine weiteren Informationen bestehen und nur ein Pfad von einer willkürlichen Menge an Pfaden verwendet wird. Beispielsweise wenn Bild 1 und Bild 3 identisch wären und wir das Bild 2 verworfen hätten, das einen komplett anderen Sachverhalt darstellt, hätten wir keine Möglichkeit, an diese Informationen zu gelangen. Dass bei der Interpolation die Bilder 1, 2 und 3 identisch sein würden ist aber ein Trugschluss. Dieser Effekt tritt insbesondere bei schnellen Veränderungen sowie einer geringen Bildwiederholungsrate ein.

Die folgende Abbildung zeigte eine mittlere Verunreinigung des Bildes:

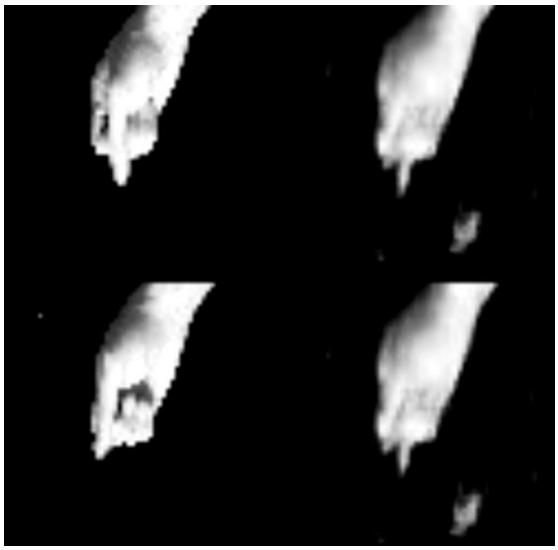


Abbildung 24: Bildverunreinigung
Quelle: Eigene Darstellung

5.4.3 Ergebnisse

Einige kritische interessante Höhepunkte des Videos, deren es einer genauen Beobachtung bedarf, sind die folgenden:

- 3:01 – Bewegung des Daumens bei vertikaler Fauststellung.
- 4:23 – Bewegung des Daumens bei horizontaler Fauststellung.
- 4:31 – Auf- und Abwärtsbewegungen der Faust.
- 6:10 – Bewegungen der unteren Fingerglieder.
- 6:55 – Diverse Auf- und Abwärtsbewegungen der Hand.
- 7:13 – Position- und Winkelveränderung der Hand.
- 8:05 – Öffnen und Schließen der ausgestreckten Hand.
- 8:33 – Schnelle Handbewegungen.
- 9:45 – Daumenbewegungen und einzelne Fingerbewegungen.

In der folgenden Tabelle sind die einzelnen Beobachtungen der jeweiligen VAE-Modelle aufgelistet:

VAE-Nr.	Beobachtungen:
2	• Die Unschärfe hält sich in Grenzen. • Geringes Flimmern. • Sehr wenige groteske Veränderungen. • Unterschied der Bildwiederholungsrate des Videos kaum wahrnehmbar. • Sämtliche Bewegungen wurden richtig erfasst.
3	• Die Unschärfe hält sich in Grenzen. • Geringes Flimmern. • Sehr wenige groteske Veränderungen.

	• Unterschied der Bildwiederholungsrate des Videos wahrnehmbar. • Ein paar Bewegungen wurden nicht richtig erfasst.
4	• Die Unschärfe ist deutlicher wahrnehmbar als in vorherigen Versuchen. • Wenig Flimmern. • Groteske Veränderungen sind deutlich häufiger als zuvor. • Unterschied der Bildwiederholungsrate des Videos deutlich wahrnehmbar. • Einige Bewegungen wurden nicht richtig erfasst bzw. lassen sich nicht eindeutig zuordnen. • Besondere Notiz: Bei 7:19 wird eine ausgestreckte Hand mit einer Faust verwechselt. Es macht den Eindruck, als würden zu der Faust ein paar Finger hinzugefügt werden, da die Formen sich zu ähneln scheinen.
5	• Besondere Notiz: Am Anfang des Videos entstehen oft die falschen Handgesten. Es scheint, als ob die Mannigfaltigkeit eine komplett andere Struktur zu haben scheint als die anderen Modele. • Geringes Flimmern. • Bewegungen werden teilweise falsch erfasst. • Teilweise groteske Veränderungen treten häufig ein. • Die Daumenbewegungen werden korrekt erfasst. • Einige Handgesten sind nicht mehr eindeutig zuordbar. • Unterschied der Bildwiederholungsrate des Videos ist deutlich spürbar. allerdings werden teilweise falsche Bilder gezeigt. Es ist nicht bekannt. ob die Ursache die Interpolation ist oder ob der Generator falsch liegt. • Es handelt sich bei dem Video keinesfalls um eine Optimierung. Es werden zwar einige Handgesten flüssiger dargestellt, allerdings stehen diese aber in keinem akzeptablen Verhältnis zu der Anzahl der falsch gezeigten Bilder.
6	• Deutliche Verbesserung gegenüber dem Modell-Nr. 5. • Normale Unschärfe feststellbar. • Wenig Flimmern. • Groteske Veränderungen sind seltener als zuvor. • Unterschied der Bildwiederholungsrate, des Videos deutlich wahrnehmbar. • Es werden nahezu alle Gesten richtig erfasst.
7	• Leichtes Flimmern. • Teilweise Unschärfe und nicht direkt zuordnungsbare Bilder. • Sämtliche Gesten wurden richtig erfasst. • Deutliche verbesserte Bildwiederholungsrate die Verläufe werden richtig eingeschätzt.
8	• Wenige unscharfe, verrauschte Bilder. • Es entstehen ein paar nicht zuzuordnende Bilder. • Es entsteht ein deutlich flüssigeres Video mit einem meist klaren Verlauf. • Kaum Flimmern. • Nahezu sämtliche Bewegungen wurden richtig erfasst.
9	• Teilweise unscharfe nicht zuzuordnende Bilder. • Relativ häufiges Flimmern. • Nahezu alle Bilder wurden richtig erfasst. • Es entsteht ein deutlich flüssigeres Video mit einem meist klaren Verlauf. • Die Interpolationsschritte sehen den ursprünglichen Bildern erstaunlich ähnlich.

10	• Teilweise unscharfe nicht zuzuordnende Bilder. • Kaum Flimmern. • Nahezu alle Handgesten wurden richtig erfasst. • Es entsteht ein deutlich flüssigeres Video mit einem meist klaren Verlauf. • Die Interpolationsschritte sehen den ursprünglichen Bildern erstaunlich ähnlich. • Besondere Notiz: Schöner Bewegungsfluss an der Stelle 7:18.

Tabelle 5: Beobachtungen der einzelnen Modelle

Quelle: Eigene Daten

Generell lässt sich eine Unschärfe in den einzelnen Videos feststellen. Allerdings entstehen in Kombination mit dem falschen Erfassen von Gesten teilweise nicht zuzuordnende Bilder. Es ist erstaunlich, dass diese Effekte so stark von den einzelnen Modellen abweichen. Prinzipiell wurden die Modelle nur im geringen Maße trainiert. Bei einem längeren Training sollten diese Effekte deutlich reduziert werden. Anders sieht es allerdings bei einer falschen Mannigfaltigkeit aus, diese nachträglich zu korrigieren würde sich meiner Meinung nach als deutlich schwieriger erweisen. Besonders auffällig scheint das Modell Nr. 5 zu sein. Die Mannigfaltigkeit scheint sich grundlegend von den anderen zu unterscheiden. Ein Blick auf den latenten Plot, wie beispielsweise latentplot_5_z25_e100, zeigt sehr häufig redundante Informationen. Es besteht die Möglichkeit dass diese redundanten Informationen im Verlauf des Trainingsprozess weniger werden. Die letzten beide Modelle veranschaulichen allerdings, dass es sehr wohl möglich ist, die Bildwiederholungsrate künstlich zu erhöhen für den Preis eines unschärferen Videos. Die Performance der einzelnen Modelle scheinen lassen sich anhand der Performancemessungen bereits erahnen, da beispielsweise das Model Nr. 5 den schlechtesten Performancewert liefert.

5.5 **Fazit des Experiments**

Wir haben gesehen, dass es sehr wohl möglich ist, die Bildwiederholungsrate künstlich zu erhöhen. Dies setzt allerdings voraus, dass eine geeignete Struktur der Mannigfaltigkeit gelernt wird. Für das Lernen der Mannigfaltigkeit ist der VAE allerdings prädestiniert und stellt meiner Meinung nach das Haupteinsatzgebiet des VAE dar. Ich denke, dass die Nachteile des VAE den Vorteilen überwiegen und würde von der Verwendung abraten.

Das Projekt würde ich dennoch als erfolgreich bezeichnen, da nur relativ wenig Trainingsproben ausreichend sind, um vorzeigbare Ergebnisse zu erzielen. Allerdings lässt sich über die realen Anwendungsgebiete streiten, da sich das Video nur auf den Aspekt der Hand beschränkt. Würden andere Elemente, verschiedene Hintergründe und diese in einer unterschiedlichen Häufigkeit gezeigt werden, hätte der VAE deutlich mehr Probleme. Das gesamte Experiment beschränkt sich nur auf ein relativ niedrig aufgelöstes Bild. Es ist nicht klar, ob sich die

Beobachtungen auch auf ein hochaufgelöstes Video übertragen lässt. Meiner Meinung nach wäre eine Lösung, wenn wir ein Video mit einer sehr geringen Bildwiederholungsrate hätten, dass wir ein weiteres Video aufnehmen, das wir zum Lernen der Mannigfaltigkeit verwenden können. Die Kombination von einzelnen Datenpunkten und einer detaillierten Mannigfaltigkeit würde zu hervorragenden Ergebnissen führen. Mit einer besseren Interpolationsmethode würden sich vermutlich auch Videos mit einer deutlich geringeren Bildwiederholungsrate restaurieren lassen, indem wir einen geeigneten Pfad auf der Mannigfaltigkeit finden.

6 Ausblick

Ich halte das maschinelle Lernen und insbesondere das ungestützte Lernen für ein Forschungsgebiet mit weitreichender Zukunft. Die Menge an produzierten Daten wird in der Zukunft deutlich zunehmen, insbesondere durch die wachsende Anzahl an Computern und dem „Internet of things“. Die Technologie, um diese Datenmengen auszuwerten, ist aber meiner Meinung nach allerdings noch nicht vollständig ausgereift. Viele der Modelle des maschinellen Lernens leben von der schieren Fülle an Daten. Dies ist meiner Meinung nach allerdings der falsche Ansatz, da beispielsweise Menschen durchaus in der Lage sind, aus vergleichbar minimalen Informationen trotzdem noch geeignete Informationen zu ziehen. Es existieren anscheinend noch nicht entdeckte Algorithmen die deutlich leistungsfähiger als die heutigen sind.

Die Auswirkungen dieser neuen Technologien werden die Welt grundlegend verändern, sei es die Arbeit, die Kommunikation sowie die sozialen Systeme. Allerdings werden diese neuen Technologien sich auch in einer emotionalen, noch nicht einschätzbaren Weise auf die Menschen auswirken. Ich denke, dass die Arbeit sehr an das Wohlergehen des Menschen geknüpft ist und sehe daher sowohl Potenzial für herausragende positive Dinge für die Menschheit, wie eine verbesserte medizinische Versorgung, sowie auch die negativen Potenziale wie ein Machtungleichgewicht zwischen der Bevölkerung und einzelnen Individuen.

Abschließend lässt sich sagen, dass, auch wenn in der Zukunft der VAE mit einer anderen Technologie abgesetzt wird, es sich um eine herausragende Technologie handelt, die einen weiten Anwendungsspielraum besitzt.

Literaturverzeichnis

Aphex34 (2015): Typical CNN, Online im Internet: https://commons.wikimedia.org/wiki/File:Typical_cnn.png, abgerufen am: 4.12.2018.

Blei, D. M./ Kucukelbir, A./ McAuliffe, J. D. (2018): Variational Inference: A Review for Statisticians, Online im Internet: https://arxiv.org/pdf/1601.00670.pdf, abgerufen am: 3.12.2018.

Charles H. Martin (2017): Free Energy and variational inference, Online im Internet: https://calculatedcontent.com/2017/09/06/free-energies-and-variational-inference/, abgerufen am: 4.12.2018.

Chervinskii (2015): Autoencoder structure, Online im Internet https://commons.wikimedia.org/wiki/File:Autoencoder_structure.png, abgerufen am: 4.12.2018.

Dumoulin, V./ Visin, F. (2016): A guide to convolution arithmetic for deep learning, Online im Internet: https://arxiv.org/pdf/1603.07285v1.pdf, abgerufen am: 9.12.2018.

Goodfellow, I./ Bengio, Y./ Courville, A. (2016): Deep Learning, The MIT Press, Online im internet: http://www.deeplearningbook.org, abgerufen am: 21.9.2018.

Higgins, I. et al. (2016): Early Visual Concept Learning with Unsupervised Deep Learning, Online im Internet: https://arxiv.org/pdf/1606.05579.pdf, abgerufen am: 4.12.2018.

Ioffe, S./ Szegedy, C. (2015): Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Online im Internet: https://arxiv.org/pdf/1502.03167.pdf, abgerufen am: 9.12.2018.

Kingma, D. P./ Welling, M. (2013): Auro-Encoding Variational Bayes, Online im Internet: https://arxiv.org/pdf/1312.6114.pdf, abgerufen am: 4.12.2018.

Kriesel, D. (2007): A Brief Introduction to Neural networks, Online im internet: http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf, abgerufen am: 22.9.2018

MacKay, D. J. C. (2005): Information Theory, Inference, and Learning Algorithms, 7.2 Version, Cambridge University Press, Online im Internet: http://www.inference.org.uk/itprnn/book.pdf, abgerufen am: 22.9.2018.

McCulloch, W. S./ Pitts W. H. (1943): A logical calculus of the ideas immanent in nervous activity, in: Bulletin of Mathematical Biophysics, Band 5, S. 115-133.

Michael Plotke (2013): 3D Convolution Animation, Online im Internet: https://commons.wikimedia.org/wiki/File:3D_Convolution_Animation.gif, abgerufen am: 4.12.2018.

Mitchell, Tom M. (1997): Machine Learning, McGraw-Hill Science/Engineering/Math.

Raschka, S. (2017): Machine Learning mit Python: Das Praxis-Handbuch für Data Science, Predictive Analytics und Deep Learning, 1. Auflage, Frechen, mitp.

Rey, G. D./ Wender K. F. (2011): Neuronale Netze: Eine Einführung in die Grundlagen, Anwendungen und Datenauswertung, 2. Auflage, Bern, Huber.

Salimans, T./ Kingma, D. P./ Welling, M. (2014): Markov Chain Monte carlo and Variational Inference: Bridging the Gap, Online im Internet: http://proceedings.mlr.press/v37/salimans15.pdf, abgerufen am: 4.12.2018.

Digitaler Anhang

Im Digitalen Anhang befindet sich folgende Ordnerstruktur:

```
DigitalerAnhang/
    README.txt
    Ergebnisse/
              [Experiment_Nummer]/
                        GeneratedImage/
                              [die latenten Plots sowie das Rekonstruktionsbild]
                        Morphed/
                              [Video des Experiments]
                        TextSaves/
                              [enthält die cost.txt und eine embeddings-Datei]
               [Die Ordner enthalten die Ergebnisse der Experimente]
              Komplett/
                        Manifold/
                              [Bilder der Manifold]
                        TextSaves/
                              [enthält die embeddings-Datei]
    ExperimentTemplate/
              Config.py
              DatasetOperations.py
              Experiment.py
              ImageHandler.py
              Model.py
              TextHandler.py
              VideoHandler.py
              Dataset/
                     GenerateDataset.py
                     OriginalVideo.avi
              embeddingModel/
                     CreateEmbeddings.py
              GeneratedImage/
              model/
              Morphed/
              TextSaves/
```

Die README-Datei enthält die Anweisungen zur Durchführung der Experimente. Außerdem enthält sie eine Anleitung zur Erstellung der Mannigfaltigkeit der einzelnen Test Ergebnisse.

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt und mich anderer als der in den beigefügten Verzeichnissen angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Der Durchführung einer elektronischen Plagiatsprüfung stimme ich hiermit zu. Die eingereichte elektronische Fassung der Arbeit entspricht der eingereichten schriftlichen Fassung exakt. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen hat.

Ort, den Vorname, Name