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:

Internationales Management

4. Fachsemester, Sommersemester 2013

Abgabedatum 21. Mai 2013

Inhaltsverzeichnis

Abkürzungsverzeichnis

ABl. ..... Amtsblatt der Europäischen Union

APU ..... Auxiliary Power Units

BIP ..... Bruttoinlandsprodukt

BMU ..... Bundesministerium für Umwelt

DEHSt ..... Deutsche Emissionshandelsstelle

EEX ..... European Energy Exchange

EG ..... Europäische Gemeinschaft

ETS ..... Emission Trading System

EU ..... Europäische Union

EUA ..... European Union Allowances

FCOM ..... Flight Crew Operating Manual

IATA ..... International Air Transport Association

ICAO ..... International Civil Aviation Organisation

IPCC ..... Intergovernmental Panel on Climate Change

LCC ..... Low Cost Carrier

OECD ..... Organisation für wirtschaftliche Zusammenarbeit und Entwicklung

TEHG ..... Treibhausgas-Emissionshandelsgesetz

UBA ..... Umweltbundesamt

Abbildungsverzeichnis

1 Einleitung

// Wird später erstellt

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 ist 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" kommt daher, 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: Bei dem unüberwachten lernen wird dem Algorithmus kein entsprechendes Zielattribut vorgegeben. Es geht überwiegend darum verborgenes Wissen bzw. Information 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 der 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 vorheri-

[1] Mitchell (1997) Seite 2
[2] Goodfellow 82

gen Handlungen in einer Umgebung. Ziel ist es, basierend auf dem Muster von Inputreizen entsprechende Aktionen, bzw. Reaktionen auszuführen mit dem Ziel seine Belohnungen (engl. reward) zu maximieren. Der Algorithmus lernt mit fortschreitenden der 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 lernt der Algorithmus das gewünschte Verhalten. Zusammengefasst lässt sich also sagen das wir Erfahrungen bzw. Daten brauchen, ein Model bzw. einen Algorithmus, der diese Daten verarbeitet und eine Performance Funktion w.r.t. der Aufgabe.

2.1 **Modelle des Überwachten Lernens**

Ziel des überwachten lernen ist es ein Model zu finden das in der Lage ist, auf neue noch nicht gesehene Daten richtig zu reagieren[4]. Damit ein Model 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 Model erhält eine Eingabe x und generiert dazu den geschätzten Wert für das Zielattribut y.

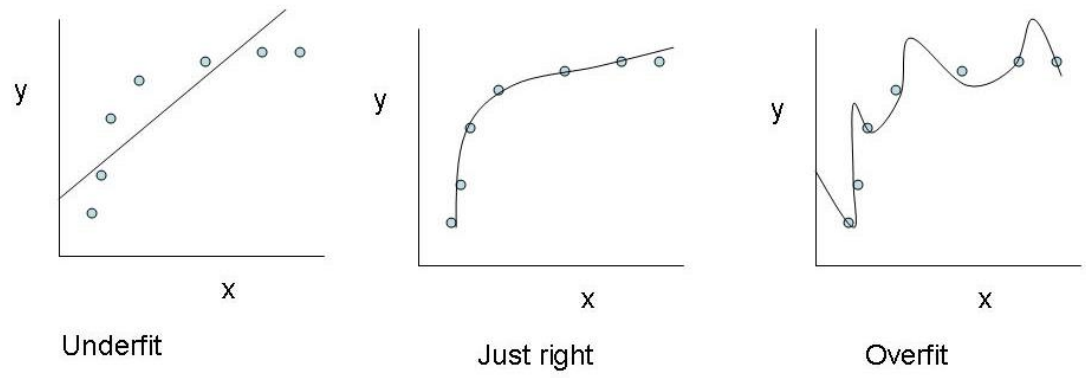


Abbildung 1: Unter- und Überanpassung anhand eines Regressionsproblems
Quelle: https://gigadom.files.wordpress.com/2014/01/41.jpg
// https://gigadom.in/2014/01/03/simplifying-machine-learning-bias-variance-regularization-and-odd-facts-part-4/

[3] David Kriesel 53
[4] Goodfellow 110

Es ist sehr deutlich erkennbar das der Graph auf der linken Seite die ursprünglichen Daten suboptimal abbildet. Das liegt daran, dass das Model versucht eine nicht lineare Funktion mit einer linearen Funktion zu approximieren. Allerdings liegt der Fehler nicht an dem Model, sondern an dem Anwender, der den Daten einen linearen Verlauf unterstellt und das Model 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. Das Bild in der Mitte zeigt ein Model, dass eine Funktion gelernt hat das 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 das die Komplexität der Aufgabe ungefähr der Komplexität des Models entsprechen soll.[5] Um die Fähigkeit des Generalisierens zu testen werden in der Praxis häufig gezielt Datensätze entfernt bzw. dem Model vorenthalten um das Model auf genau diese Eigenschaften hin zu überprüfen. Diese Überprüfung kann beispielsweise durch die Performance Messung stattfinden, sofern die Labels bekannt, um einen Fehlerwert zu berechnen und diese miteinander zu vergleichen.

2.2 **Algorithmen des Überwachten Lernens**

Es existieren eine Vielzahl an Modellen um die Aufgabestellung des überwachten Lernens zu lösen. Je nach Menge der Daten, der Problemstellung sowie Verfügbarkeit von Rechenressourcen hängt die Wahl des entsprechenden Algorithmus 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 Vielfältige 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 in diesem Themenfeld gehören Warren McCul-

[5] Goodfellow S.112
[6] Vgl. Goodfellow et al (2016), S. 224

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

Die Idee hinter neuronalen Netzen ist, dass ein Neuron durch die eine Reihe von Inputs aktiviert wird bzw. Informationen erhält und 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.

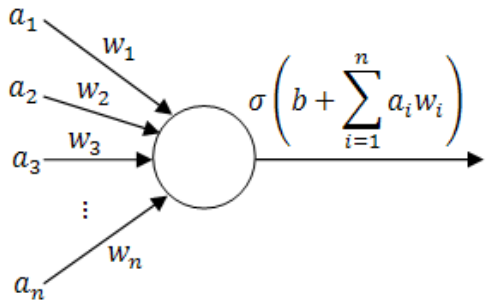


Abbildung 2: Neuronales Netz mit einem einzigen Neuron

Die Abbildung zeigt den schematischen Aufbau 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 $a \in \mathbb{R}^n$ (a entspricht in dem Falle des Eingabevektors x) 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 b hinzuaddiert da der vorhergesagte Graph sonst nur durch den Ursprung laufen würde[8]. Beispielswiese könnte die Funktion $f(x) = 1$ nicht abgebildet werden ohne den Bias Parameter. Abschließend wird noch eine sogenannte Aktivierungsfunktion angewandt um nicht lineare Funktionen abzubilden 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 zu versetzen. Wenn das Gewicht nahe bei null ist übt es keine oder nur eine geringe Wirkung aus, es hat also weder eine animierende noch hemmende Wirkung.[9]

Dieses Model ist allerdings nur beschränkt möglich eine beliebig komplexe Funktion zu erlernen deswegen wird in dem Kontext von neuronalen Netzen häufig der Begriff „Deep learning“ verwendet. Es handelt sich dabei um die Aneinanderreihung von neuronalen Netzschichten um komplexe geschachtelte Funktionen zu erlernen.

[7] Warren S. McCulloch and Wallter H Pitts (1943)
[8] Goodfellow 109
[9] Vgl. Neuronale Netze S.15

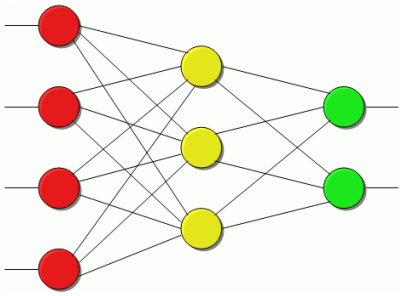


Abbildung 3: Der Aufbau eines tiefen neuronalen Netzes
Quelle: http://www.methoden-psychologie.de/neuronale_netze.html

http://www.methoden-psychologie.de/neuronale_netze.html

Die Abbildung zeigt ein tiefes neuronales Netz mit einer Inputschicht (rot), einer versteckten Schicht (gelb) und einer Ausgabeschicht (grün). Die mathematische Vorschrift um die Ausgabe zu berechnen 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 vorwärts gerichtete Propagieren bezeichnet. [10]

Das designen von versteckten Schichten wie beispielsweise die Auswahl von Aktivitätsfunktionen sowie Festlegung der Anzahl der Neuronen pro Schicht wird im Zusammenhang mit der Struktur, mit der die einzelnen Schichten in Verbindung stehen wird auch als Model Architektur bezeichnet.[11] 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). 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 die Relevanz dieser zu zeigen modifizieren wir die Funktion für die vorwärts gerichtete Propagation. In ihr wird auf die Aktivierungsfunktion verzichtet sowie auf dem Bias Parameter um die Gleichung übersichtlicher zu gestalten.

$$f(x) = f^2\big(f^1(x)\big)\, mit\, f^i(x) = \ W_i x$$

Anders formuliert entsteht:

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

[10] NN 52
[11] Goodfellow 197

Das tiefe neuronale Netz hat also nicht die Fähigkeit komplexere Funktion zu erlernen, da sie mit einer einzigen Gewichtsmatrix W* ausgetauscht werden kann. Es werden also zwingend Aktivierungsfunktionen benötigt um nicht lineare Funktionen zu erlernen.[12]

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, die biologische Plausibilität oder aber auch die Differenzierbarkeit der Funktion.[13] Eine weitere interessante Eigenschaft ist wie sich die Ableitung der Funktion verhält, also ob Gradient auf bestimmten Teilbereichen der Funktion sehr klein ist oder sehr groß ist.[14] Sollte der Gradient klein sein macht diese Eigenschaft das Lernen für den Algorithmus schwerer und es wird mehr Zeit benötigt um ähnliche Ergebnissen zu erzielen wie bei anderen Aktivierungsfunktionen. Nachfolgend sind einige gängige Aktivierungsfunktionen aufgelistet.

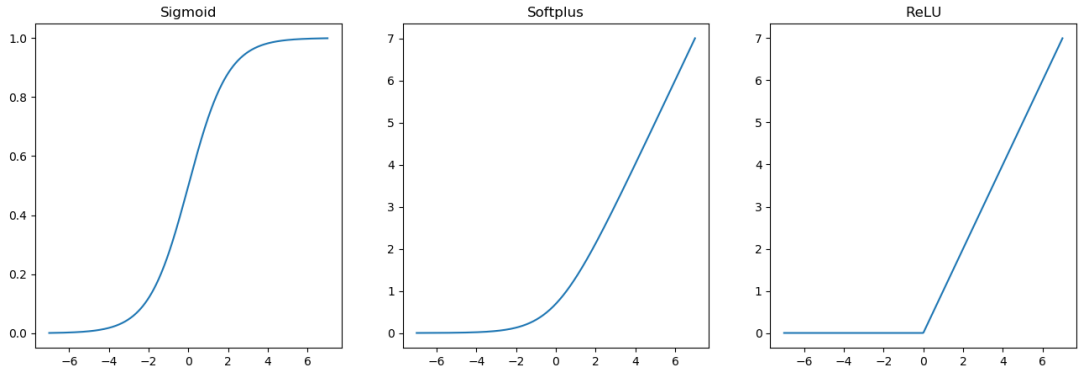


Abbildung 4: Gängige Aktivierungsfunktion. Links die Sigmoid Funktion in der Mitte die Softplus Funktion und rechts die ReLU Funktion

Quelle: Eigener Plot

- 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 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.

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

[12] Goodfellow 172
[13] NN 22
[14] Goodfellow 179

- 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 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 Steigung konstant bleibt.[15]

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

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[16]. Vergleichbar wie bei einem tieferen neuronalen Netz ist das Netzwerk auch in verschiedenen Schichten aufgeteilt. Jede dieser Schichten besteht typischerweise aus:

- einer Konvolutionsoperation
- einer Aktivierungsfunktion
- und einer Poolingsoperation

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 dessen Input die Ausgabe der letzten Konvolutionsschicht darstellt die als Vektor neu arrangiert worden sind.

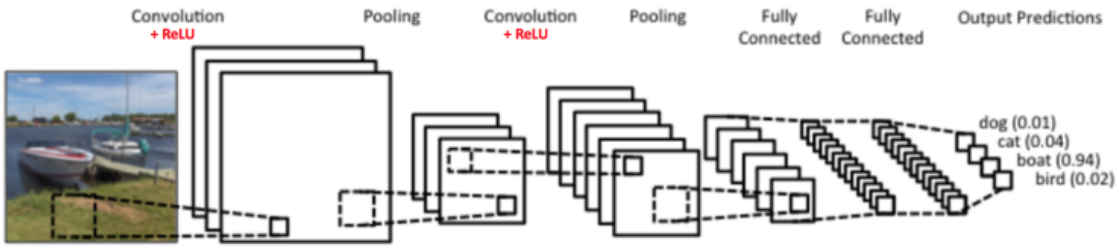


Abbildung 5: Schematischer Aufbau eines Konvolutionalen Netzwerkes
Quelle: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[15] Goodfellow S 192
[16] Bishop S 267

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 Information des Bildes enthält. Der Filter ist üblicherweise quadratischer Natur und umfasst n^2 Gewichte. In der Vergangenheit wurden viele dieser Filter per Hand entworfen um bestimmte Effekte bei der digitalen Bildverarbeitung zu erzielen, wie einen Glättungseffekt oder Kantenerfassung um beispielsweise abrupte Änderungen in dem Bild aufzuspüren. Allerdings anstelle 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 Stell i,j zu berechnen lautet:

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

$S(i,j)$ 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 die nächste Operation berechnet werden soll.

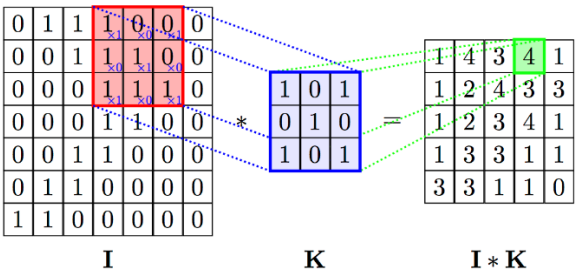


Abbildung 6 Konvolutionsoperation

Quelle: https://github.com/PetarV-/TikZ/tree/master/2D%20Convolution

Würde in der Praxis nur einen 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 von der Anzahl der Filter. Die erste Matrix, das ursprüngliche Bild, ist häufig auch ein Array an Matrizen da die einzelnen

Kanäle für den rot, grün und blauen Farbton separat gespeichert werden. Im Falle eines Graubildes handelt es sich um ein allerdings um eine einfache Matrix.
Nachdem jede Konvolutionsoperation in der Schicht vollzogen wurde wird auf jeden berechneten Wert ein Bias Parameter hinzuaddiert und eine Aktivierungsfunktion auf die berechneten Werte angewendet.
Eine weitere Operation ist die Poolingoperation. In ihr wird auf den zuvor berechneten Schritt ein weiterer Filter angewendet, der die Werte gegenüber kleinen Änderungen weniger anfällig machen soll. [17] Die Poolingoperation erreicht dieses Verhalten dadurch, dass der Filter je nach Wahl den Mittelwert oder den Maximalwert zurückgibt.

2.3 **Lernen**

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

2.3.1 Performancemessungen

In vielen Bereichen des maschinellen Lernens werden den Daten ein bestimmtes statistische Model unterstellt das diese Daten generiert, dem sogenannten Daten generierenden Prozess P_{Data}.[18] 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:

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

[17] Goodfellow 357
[18] Goodfellow 111

Sämtliche Elemente des Differenzvektors zwischen x und y werden potenziert und aufsummiert. In der Literatur wird diese Metrik gerne ohne Herleitung eingeführt und dem Leser eine intuitive Erklärung gegeben. Also das 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 verschiedener 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 das sie keiner negativen Werte annehmen kann und das 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.

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: [19]

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

‚y‘ entspricht dabei dem richtigen Wert, der in Relation zu dem x steht. Je genauer unsere Vorhersage von μ mit gegebenen 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

[19] Goodfellow 132

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 Tatsache 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:

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

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 der die Hauptaufgabe der Performancemessung daraus zwischen zwei verschiedenen Algorithmen den besten auswählen zu können. Die Aussagekraft der verschiedenen Metriken sind aber mehr oder weniger abstrakt und geben 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 ein „Spam-Flag" für eine E-Mail zu setzen. Allerdings würden die Performancemessung trotzdem einen Fehlerwert enthalten da wir eigentlich für diese E-Mail einen 100 Prozentige Genauigkeit anstreben. Deswegen ist es in der Praxis häufig sinnvoll alternative Metriken zu erstellen die eine Aussagekraft besitzen die für Menschen nachvollziehbar ist. Die Marketing Abteilung wäre beispielsweise an dem Verhältnis interessiert das eine E-Mail eine falsch zugeordnet 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, um die Likelihood zu berechnen, 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. Das ist möglich, da der Wert von θ_{ML} bei beiden Funktionen identisch bleibt.[20]

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

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

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.[21] 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

[20] Goodfellow 146
[21] Goodfellow 98

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 den 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 Outputunit berechnet und anschließend schrittweise für die einzelnen versteckten Schichten.

Der Backpropagation-Algorithmus lässt sich in fünf verschiede Schritte einteilen die nötig sind um eine Iteration des Lernprozesses auszuführen.[22] Diese Schritte werden so lange wiederholt, bis der Algorithmus konvergiert oder eine maximale Anzahl von Schritten erreicht wurde.

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 dem das Neuron i in Beziehung steht und die Informationen des Neurons 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.

[22] Mitchell 98

4. Berechne die Veränderungen der 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 kann dabei über die Kettenregel der Differentialrechnung erfolgen.[23] 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 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 vereinfacht 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:

[23] Mitchell 101 Derivation

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

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

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.[24] Aufgrund dieser Flexibilität für verschiedene Problemstellungen und der 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 überspringt und hoffentlich ein besseres lokales Minimum anstreben. Häufig eingesetzte Verfahren in der Praxis sind beispielsweise Lernparameter die sich mit der Zeit verringern oder auch Verfahren die das Momentum des Gradienten nutzen sowie Verfahren die eine statistische Verteilungen der Aktivierungen der Neuronen betrachten.

Graphisch kann das folgendermaßen veranschaulicht werden:

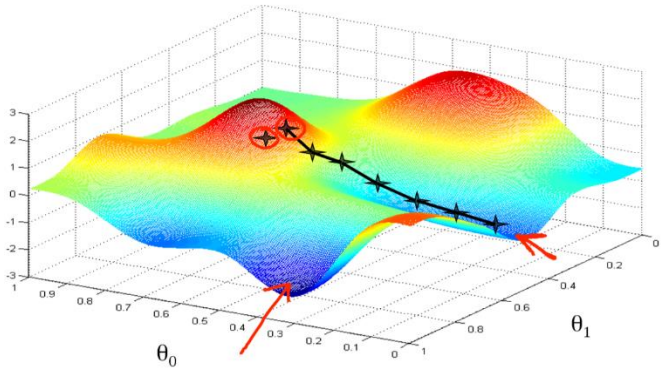


Abbildung 7: Gradientenabstiegsverfahren
Quelle: https://hackernoon.com/gradient-descent-aynk-7cbe95a778da

Die Hügellandschaft zeigt die Bereiche, an denen die Performancefunktion groß bzw. klein ist. Der schwarze Strich zeigt die einzelnen Stationen des Gradientenabstiegsverfahrens an bzw.

[24] Goodfellow 205

die Verschiebung der Position der Parameter θ_0 und θ_1. In dieser Abbildung wird der Algorithmus lediglich von diesen beiden 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 eine nicht konvexes Optimierungsproblem. Die Gründe dafür im Fall von Neuronalen Netzen sind vielfältig. Beispielsweise durch eine sogenannte Gewichts-Asymmetrie, da die einzelnen Gewichte der Neuronen innerhalb einer Schicht ausgetauscht werden können, da die Reihenfolge der Neuronen keine Rolle spielt. Ein weiteres Optimierungsproblem sind Sattelpunkte. Mathematisch lässt es sich zeigen, dass das Verhältnis zwischen Sattelpunkten und lokalen Optima in einem hochdimensionalen Raum exponentiell wächst.[25] Es ist wahrscheinlicher an einen Sattelpunkt zu geraten als einem lokalen Optimum. Prinzipiell sind Sattelpunkt 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.

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

[25] Goodfellow 285 f

3 Inferenz

Unter Inferenz (lat. „Inferre“) wird das Schlussfolgern von Sachverhalten verstanden. Im Wesentlichen geht es dabei 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 das Sokrates lebt sowie ein Mensch ist wäre eine Schlussfolgerung das 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 von Fragestellung lässt sich durch das observieren von vergangenen Ereignissen beantworten. Es lässt die Wahrscheinlichkeit ermitteln, mit der die Aussage zutrifft bzw. nicht zutrifft. Die zweite Art von 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 Inferenz beschäftigen, die auf diese Art von Problemstellungen angewendet werden kann. Wir werden uns zunächst mit dem Bayes Theorem vertraut machen mit der Inferenz durchgeführt werden kann sowie dem Problem mit dieser Methode. Anschließend werden wir im Detail den Expectation-Maximization-Algorithmus der das Fundament für Variational Inferenz 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 x Berech-

nen. 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 Fragestellungen ein guter erster Schritt zur Lösung des Problems ist, je nachdem ob das eine oder andere einfacher zu berechnen ist.

Eine Möglichkeit um eine Intuition über das Bayes Theorem erhalten ist durch das studieren eines Beispiels[26]. Angenommen wir wären von unserem Arzt auf eine gefährliche Krankheit getestet und dieser Test wäre positiv ausgefallen. Der Arzt gibt den Hinweis das die Genauigkeit des Tests bei 99% liegt. Wie besorgt wären wir? Anhand von Statistiken erfahren wir das ca. 1 von 1 000 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 das wir eine ernsthafte Krankheit haben. Im Verhältnis zu 99% eine deutliche Besserung. Der Schlüssel dieser Verzerrung liegt bei der zusätzlichen Information zu der Auftretungswahrscheinlichkeit. Beispielsweise haben wir bei obigen Zahlen circa 10 Falschklassifizierungen. Die Chance, dass wir die Falschklassifizierung sind oder der Test wahrhaftig positiv ist drückt das Bayes Theorem in diesem Beispiel aus.

[26] Mackkay 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[27] um Inferenz durchzuführen. In der Literatur haben sich folgende Verfahren etabliert:

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

Wir werden uns nur mit der kompletten Enumeration sowie der Variational Inferenz befassen. Es sollte allerdings erwähnt werden, dass es 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 Theorem haben, oder diese Berechnung durchführen können. Der Vorteil ist das wir exakte Ergebnisse erhalten allerdings durch einen oftmals sehr hohen Rechenaufwand. Dieser Rechenaufwand ist teilweise so hoch das 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 Anzahl der Daten bezieht die nötig sind um eine möglichst genaue Inferenz durchzuführen.[28]

Eine Möglichkeit um dieses Problem zu lösen sind sogenannte approximierte Methoden anzuwenden. Variational Inferenz gehört zu dieser Rubrik und wir werden diese anhand des Expectation-Maximization-Algorithmus herleiten.

[27] MacKay 294
[28] Bishop 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 variationale Bayesian Inferenz (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.

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.[29] Dieser Prozess wiederholt sich solange bis keine neuen Clusterzuweisungen mehr durchgeführt werden. Diese zwei Schritte können als einen Erwartungsschritt und einen Optimierungsschritt aufgefasst werden.

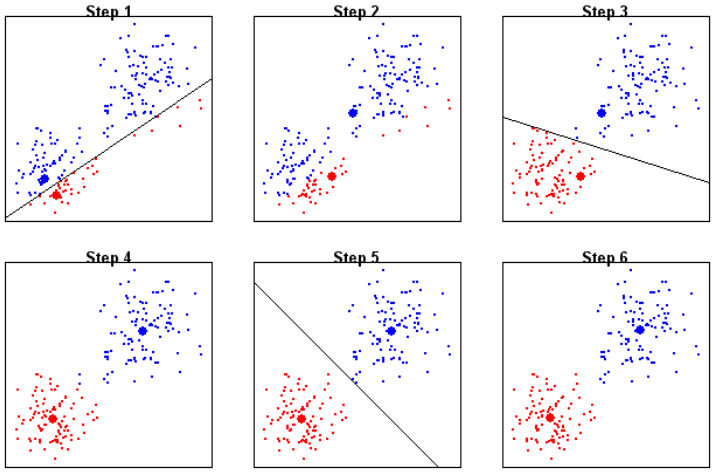


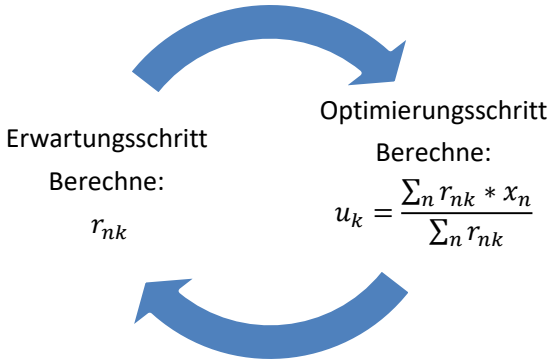
Abbildung 8: K-Means Algorithmus
Quelle: https://medium.com/@dilekamadushan/introduction-to-k-means-clustering-7c0ebc997e00

Die Abbildung zeigt den Verlauf der Clusterbildung und jedes Bild beschreibt dabei einen Zyklus in dem EM Algorithmus Schritt. 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 Cluster Ursprung darstellt. Die Zugehörigkeit sowie die Clustermittelpunkte sind vorab unbekannt und stellen die gesuchten Werte des Algorithmus dar. Formal

[29] Bishop 424 ff.

lässt sich das folgendermaßen ausdrücken. Dabei stellt r_{nk} die Zugehörigkeit von dem Datensatz n zu dem Cluster k dar.

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



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 das 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 Performance Funktion, 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. [30]Dabei wird anstelle einer harten Clusterzuweisung eine weiche Clusterzuweisung gefunden, die 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 Algorith-

[30] Bishop 439 f

mus 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 Wahrscheinlichkeiten, 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'te Datensatz das Attribut j.

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

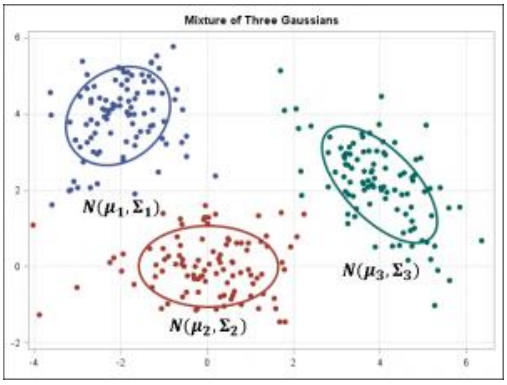


Abbildung 9: Clustering mit einer Gaußschen Mischverteilung
Quelle: https://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/

3.3.3 Herleitung der Variational Inferenz

Wie zum Anfang beschrieben stellt der Expectation-Maximization Algorithmus das Herz von Variational Inferenz Methoden dar. Wir wollen uns daher noch das generelle Konzept betrachten sowie die Herleitung.[31]

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 an 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-

[31] https://arxiv.org/pdf/1601.00670.pdf S .6f

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 das $\mathbb{E}[\log(\mathrm{P(x)}] = \log P(x)$ ist, da sie unabhängig von dem Erwartungswert von z ist ergibt sich:

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

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

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 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:

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

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:

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

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

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

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, dass wir zwei Parameter haben. Einen neuen gesuchten Parameter θ^{new} und einen alten Parameter θ^{old} die unsere Dichtefunktion beschreibt. [32]Wenn wir für $q(z) = P(z|x, \theta^{\text{old}})$ einsetzen bzw. unserem aktuell besten Model können wir auf folgende Gleichung kommen:

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

$$= \sum_z P(z|x, \theta^{\text{old}}) * \log P(z,x) - \sum_z P(z|x, \theta^{\text{old}}) * \log P(z|x, \theta^{\text{old}})$$

$$= \sum_z P(z|x, \theta^{\text{old}}) * \log P(z,x) + Const$$

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

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

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 von dem vergangenen Kapitel entstehen. Im Falle von einer Normalverteilung handelt es sich 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:

[32] Bishop 452

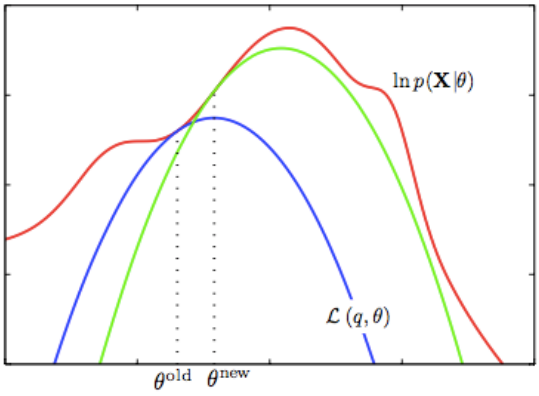


Abbildung 10: Variational Inferenz
Quelle: https://am207.github.io/2017/wiki/EM.html

3.4 **Variational Inferenz**

Bisher haben wir uns mit dem Expectation-Maximization-Algorithmus vertraut gemacht sowie deren Herleitung. Wir haben für unsere Funktion mit dem Posterior gleichgesetzt $\mathrm{q}(\mathrm{z}) = \mathrm{P}(\mathrm{z}|\mathrm{x}, \theta^{\mathrm{old}})$ um das generelle Framework der Variational Inferenz kennenzulernen. Es ist allerdings nicht zwingend notwendig diesen Ansatz zu folgen um das Inferenz Problem 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 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 Distribution abgedeckt, die untereinander keine Abhängigkeiten haben. Dieser Ansatz wird auch als „Mean field"-Ansatz bezeichnet und gehört zu den einfacheren Varianten von Variational Inferenz. Wissenschaftler haben auch andere Dichtefunktionsfamilien untersucht die beispielsweise Abhängigkeiten untereinander aufweisen. Dieser Ansatz wird auch als strukturierte variational Inferenz bezeichnet.[33] Dieser Ansatz würde Allerdings an dieser Stelle zu weit führen. Das Model kann Aufgrund einer flexiblen Disnicht an einer Überanpassung leiden aufgrund einer zu flexiblen Distribution. Die Approximation kann dadurch nur noch genauer werden.[34]

Wie wir gesehen haben geht es bei VI darum den ELBO zu maximieren. Wenn wir in die Gleichung $q(z) = \prod_i q_i(z_i)$ erhalten wir folgendes:

[33] https://arxiv.org/pdf/1601.00670.pdf S8
[34] Bishop 464

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

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

Ein großer Vorteil, wenn die einzelnen Distributionen unabhängig sind, ist das 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.

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

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

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

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

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. Das erlaubt es VI Methoden auch auf große 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 verla-

gern 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

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.[35] Beide Szenarien sind nur bedingt optimal, da Bereiche ein- bzw. ausgegrenzt werden.

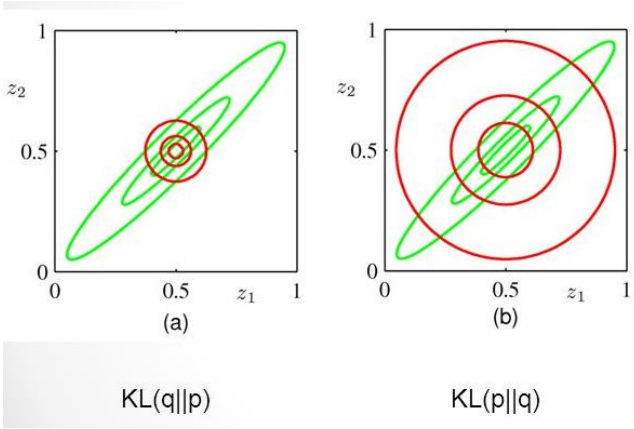


Abbildung 11 KL-Abweichung und negative KL-Abweichung
Quelle: https://images.slideplayer.com/15/4703655/slides/slide_21.jpg (oder bishop)

Eine Folge des „Mean field"-Ansatzes ist das 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.

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.

Es sollte noch erwähnt werden, dass es andere Methoden gibt um das Inferenz Problem zu lösen. Beispielsweise bietet die Monte Carlo Markov Chain 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 anzuwenden.[36]

[35] Bishop 468
[36] Tim Salimans http://proceedings.mlr.press/v37/salimans15.pdf

Autoencoder

- Mannigfaltigkeit
- Hautachsentransformation
- Autoencoder
- Variational Autoencoder

Projekt

- Datensatz
- Problemstellung
- Vorgehensweise
- …

$$Text \underbrace{\backslash.\text{underbrace } \backslash.\text{below}}_{below}$$

Literaturverzeichnis

// Wird später ausgefüllt… Es sind noch die alten Daten von dem Muster drin.

Albers, S./Peters, H. (2007): Emission Trading – Strategy Implications for Airlines, in: Association for European Transport (Hrsg.): Proceedings of the 2007 European Transport Conference, Noordwijkerhout.

Brack, D./Grubb, M., Windram, C., et al. (2000): International trade and climate change policies, London, Earthscan.

Bauchmüller M./Gammelin, C. (2010): Deutscher Streit blockiert Klimaplan/Emissionshandel für Flugzeuge auf der Kippe, in: Süddeutsche.de, 17.05.2010, Online im Internet: http://www.sueddeutsche.de/wirtschaft/emissionshandel-fuer-flugzeuge-auf-der-kippe-deutscher-streit-blockiert-klima plan-1.219438, abgerufen am 17.02.2015.

Brack, D./Grubb, M., Windram, C., et al. (2000): International trade and climate change policies, London, Earthscan.

Coase, R. H. (1960): The Problem of Social Cost, in: Journal of Law and Economics Vol. 3. (Oct. 1960), S. 1-44.

o.V. (2011): Opel kämpft um Bochum, in: Frankfurter Allgemeine Zeitung, Nr. 15, 18.03.2011, S. 5.

Europäische Union (2009): Amtsblatt der Europäischen Union ABl. L 103 vom 23.4.2009, S. 10-29 (DE), Online im Internet: http://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri= CLEX:32 00 9D0339&rid =1, abgerufen am 17.02.2015.

Fees, E. (2015): Art. Coase-Theorem, in: Gabler Wirtschaftslexikon/Das Wissen der Experten, Online im Internet: http://wirtschaftslexikon.gabler.de/Definition/coase-theorem.html, abgerufen am 17.02.2015.

International Civil Aviation Organisation (2007): ICAO Environmental Report 2007, Online im Internet: http://www . icao.int/environmental-protection/Documents/Env_Report_07.pdf, abgerufen am 18.02.2015.

Umweltbundesamt (2015): Der Europäische Emissionshandel, Online im Internet: http://www.umweltbundesamt.de/daten/klimawandel/der-europaeische-emissionshandel, abgerufen am 19.02.2015.

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