Variational Autoencoder für modelbasierte Recommender Systeme

Bachelor-Thesis

zur Erlangung des akademischen Grades B.Sc.

Daniel Ollhoff

2269742

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

Erstprüfer: Prof. Dr.-Ing. Sabine Schumann

Zweitprüfer: Prof. Dr. Kai von Luck

Hamburg, 6. 8. 2020

Inhaltsverzeichnis

1 Einleitung **6**

1.1 Thematik 6

1.2 Struktur 8

2 Grundlagen **9**

2.1 Recommender Systeme 9

2.1.1 Arten von Bewertungen 10

2.1.2 Kollaboratives Filtern 11

2.1.3 Inhaltsbasiertes Filtern 13

2.2 Latent Dirichlet Allocation 13

2.3 Künstliche neuronale Netze 17

2.3.1 Kategorien des maschinellen Lernens 20

2.3.2 Aktivierungsfunktion 21

2.3.3 Gradientenverfahren 23

2.3.4 Fehlerrückführung 24

2.3.5 Variational Autoencoder 26

2.3.5.1 Verlustfunktion eines Variational Autoencoder 29

2.3.5.2 Fehlerrückführung bei einem Variational Autoencoder 30

2.3.5.3 Variational Autoencoder Variationen 32

3 Konzept **33**

3.1 Szenario - 1: Kombination von Nutzer und Gegenstandvektor 34

3.2 Szenario - 2: Verfall von Nutzer Bewertungen 34

3.3 Szenario - 3: Nutzervektor mit vielen nullen 35

4 Implementation **36**

4.1 Technische Hilfsmittel der Umsetzung 36

4.2 Datenvorverarbeitung 37

4.3 Latent Dirihlet Allocation Implementierung und Evaluation 38
4.4 Datenbasis für den Variational Autoencoder 41
4.5 Implementierungen des Variational Autoencoder 42
4.5.1 Variational Autoencoder für die Rekonstruktion 42
4.5.2 Multinomial-VAE für das Kollaborative Filtern 43

5 Evaluierung **44**
5.1 Evaluierung des Trainingsprozesses 44
5.1.1 Variational Autoencoder für die Rekonstruktion 45
5.1.2 Variational Autoencoder für das Kollaborative Filtern 49
5.2 Evaluierung der Szenarien 51
5.2.1 Szenario - 1: Kombination von Nutzer und Gegenstandvektor . 52
5.2.2 Szenario - 2: Verfall von Nutzer Bewertungen 53
5.2.3 Szenario - 3: Nutzervektor mit vielen Nullen 54

6 Fazit **56**

Abbildungsverzeichnis **57**

Tabellenverzeichnis **59**

Literaturverzeichnis **60**

Abstract

This thesis deals with the question of the applicability of Variational Autoencoders in the context of a Recommender System. Based on textually available information, the data set is first examined by means of Latent Dirichlet Allocation and trough this, the data base for the Variational Autoencoder is created. On this basis, different variations of the Variational Autoencoder are trained and hyperparameter adjustments are evaluated. Finally, a statement about the performance and usefulness in the context of Recommender Systems is made based on the output of these Variational Autoencoders. In addition, the basics of Recommender Systems, Latent Dirichlet Allocation,Variational Autoencoders and artificial neural networks are discussed.

Recommender Systems, Collaborative Filtering, Content-based Filtering, Latent Dirichlet Allocation, Topic Modelling, Natural Language Processing, Variational Autoencoder, beta-Variational Autoencoder, Variational Autoencoders for Collaborative Filtering, Machine Learning, Artificial Neural Networks

Zusammenfassung

Diese Thesis beschäftigt sich mit der Frage der Anwendbarkeit von Variational Autoencoder im Kontext eines Recommender Systems. Auf Basis von textuell vorliegenden Informationen wird zunächst mittels Latent Dirichlet Allocation der Datenbestand untersucht und die Datenbasis für den Variational Autoencoder geschaffen. Auf dieser Basis werden verschiedene Variationen des Variational Autoencoder trainiert sowie Hyperparameter Anpassungen evaluiert. Abschließend wird anhand der Ausgabe dieser Variational Autoencoder eine Aussage über die Performanz und Sinnhaftigkeit für Recommender Systeme getroffen. Zudem werden die Grundlagen von Recommender Systeme, Latent Dirichlet Allocation,Variational Autoencoder und künstlicher neuronaler Netze erörtert.

Recommender Systeme, Kollaboratives Filtern, Inhalts-basiertes Filtern, Latent Dirichlet Allocation, Topic Modellierung, Verarbeitung natürlicher Sprache, Variational Autoencoder, beta-Variational Autoencoder, Variational Autoencoder für Kollaboratives Filtern, Maschinelles Lernen, Künstliche neuronale Netze

Häufig verwendete Abkürzungen

KNN - Künstliches neuronales Netz
VAE - Variational Autoencoder
LDA - Latent Dirichlet Allocation
ReLU - Rectified Linear Unit
SELU - Scaled Exponential Linear Unit
Tanh - Hyperbeltangens Funktion
ELBO - evidence lower bound

1 Einleitung

1.1 Thematik

Recommender Systeme, oder im Prinzip Systeme die automatisiert Empfehlungen von Inhalten generieren, filtern die immer grösser werdende Flut an verfügbaren Inhalten und Informationen für die Nutzer von, häufig webbasierten, Plattformen. Sie versuchen, für den Benutzer nur relevante Inhalte aus einem Meer von eventuell nichtigen Inhalten zur Verfügung zu platzieren. Loggt man sich zum Beispiel bei Netflix ein, gelangt man auf eine Startseite, die für jeden Abonnenten anders aussehen wird. Auf Basis der Interaktion eines Nutzers mit den Gegenständen auf Netflix oder anderen Plattformen, werden so Inhalte priorisiert oder vernachlässigt und vorlaufend an die prognostizierten Interessen eines jeden einzelnen angepasst. Oder, bewegt man sich auf Amazon, bemerkt man bestimmt den Webinhalt *Wird oft zusammengekauft*, welcher unter dem Verkaufspreis zu finden ist. Im Jahr 2006 schrieb Netflix einen eine Million Dollar Preis aus für die Entwicklung eines Systems, das die Vorhersagegenauigkeit ihres bestehenden Systems übertrifft (Netflix 2006). Zusammengefasst lässt sich also sagen, dass Empfehler Systeme versuchen Assoziationen zwischen Nutzern herzustellen und Nutzer in Gruppen zusammenzuführen die an sich die gleichen Präferenzen haben.

Die Herausforderung bei der Entwicklung solcher Systeme besteht darin einen Benutzer und die Beziehung zwischen ihm und mehreren Inhalten zu charakterisieren, um so eine bessere Filterung von Informationen bzw. Produkten gewährleisten zu können. Das Ziel ist, dass Nutzer mit ähnlichen Verhaltensmustern die gleichen Dinge mögen. Zum Beispiel hat ein Nutzer noch nicht mit einem Inhalt interagiert, gekauft oder begutachtet, einige Mitnutzer, die sich in der gleichen charakteristischen Gruppe befinden, allerdings schon, wird dieser Inhalt mit hoher Wahrscheinlichkeit diesem Nutzer ebenfalls gefalle. Bei alldem erreicht die Anzahl an zur Verfügung stehenden

Produkten nicht selten zehntausende Einträge auf größeren Plattformen. Die schiere Anzahl bringt ein weiteres Problem mit sich. Die Zeit, Rechenleistung und Speicherkapazität solche Beziehung zwischen Produkten und Nutzern oder auch Nutzer zu Nutzer numerisch auszudrücken ist immens. Zudem müssten die sich ständig wandelnden Präferenzen eines Nutzers berücksichtigt werden. Persönlich gesehen ändert sich, am Beispiel Musik oder auch Film, die Vorlieben alle paar Wochen oder Monate. Dabei kann unter anderen die Jahreszeit eine Rolle spielen oder Herausforderungen, die einem Menschen im Leben begegnen. Hinzu kommt oftmals die Problematik des Kaltstarts. Hierbei können keine Qualitativen Produktempfehlungen gemacht werden, wenn nicht genügend Informationen über einen Nutzer oder Inhalt verfügbar sind. Dies tritt auf, wenn sich ein Nutzer neu auf einer Online-Plattform anmeldet oder ein neuer Inhalt zum Inventar hinzugefügt wird.

Hier könnte eine Form der künstlichen neuronalen Netze helfen, die sich *Variational Autoencoder* nennen. Diese gehören zu einer speziellen Art, die vorranging bei Problemen, wie zum Beispiel Komprimierung von Daten oder auch der Generierung von Bildern zum Einsatz kommen. Diese Netze sind sehr gut darin die wichtigsten Eigenschaften eines Objekts selbständig zu lernen und daraus abgewandelte Objekte zu generieren oder kompromittierte Objekte wiederherzustellen. Wiederum könnte das einem *Recommender System* nutzen, um eine Nutzergruppe, Produktgruppe oder Beziehungen zu destillieren und Empfehlungen auf Basis dessen zu treffen. Durch diese Eigenständigkeit gehören die *Variational Autoencoder*, im Rahmen der unterschiedlichen Arten des maschinellen Lernens, zur Art des unüberwachten Lernens.

Diese Arbeit bewegt sich Gedanklich im Bereich von Online-Plattformen wie Amazon oder Netflix. Anzumerken ist das diese zwar die prominentesten Anwender von *Recommender Systemen* sind, es allerdings eine Vielzahl an Einsatzmöglichkeiten für *Recommender Systeme* gibt. Beispielweise Fahrassistenzsysteme oder in einer Manufaktur zur Regelung von Maschinen. Geschuldet wird dieser Umstand den für diese Arbeit zur Verfügung stehenden Daten, welche in ausreichender Menge vorhanden sein müssen um ein Neuronales Netz trainieren zu können.

1.2 Struktur

Im zweiten Kapitel werden die Grundlagen des maschinellen Lernen, *Variational Autoencoder* und *Recommender Systeme* erläutert. Darunter sind Architektur und Ablauf neuronaler Netze, die Besonderheiten von Variational Autencoder im Detail und die unterschiedlichen Arten wie Empfehlungen generiert werden. Aber auch einige Funktionen und Algorithmen, die sich in der Praxis als zuverlässig erwiesen haben, werden hervorgehoben, die wiederum in dem Code zu dieser Arbeit implementiert werden. Auf Basis dieser Grundlagen wird in Kapitel drei ein Konzept zur Lösung diverser Probleme vorgestellt. Welches in Kapitel vier Schritt für Schritt implementiert werden wird. Kapitel fünf wird sich mit der Evaluierung und Interpretation der Ergebnisse beschäftigen und darauffolgend wird diese Arbeit mit einem Fazit in Kapitel sechs abgeschlossen.

2 Grundlagen

2.1 Recommender Systeme

In diesem Unterkapitel werden Einzelheiten von Recommender Systemen erläutert sowie ihre verschiedenen für diese Arbeit relevanten Methodiken, Erscheinungsarten und Begrifflichkeiten.
Empfehler Systeme sind eine Ansammlung verschiedener Methodiken und finden am häufigsten Anwendung im Internet, beispielweise e-Shops oder Plattformen wie Amazon. Es könnte aber auch ein System in einem Auto installiert werden. Beispielhaft sind hierfür Fahrassistenzsysteme. Im Internet sind Individuen allerdings einzeln identifizierbar. Bleibt man bei dem Beispiel des Autos, wäre es nicht unbedingt klar wer dieses Auto gerade bedient. Des Weiteren lassen sich im Kontext des Internet Verhaltensweisen feingranulierter speichern und verarbeiten die für eine bestmögliche Empfehlung unabdingbar sind. Am Beispiel einer Online Plattform versuchen sie das Verhalten von Benutzern zu Kategorisieren und auf dieser Basis Vorhersagen zu treffen, um so Kundenbindung sowie Dauer der Besuchszeiten zu erhöhen und die Flut an Informationen auszusieben. Daraus resultiert meistens auch ein erhöhter Umsatz pro Kunde. Denn je schneller ein Kunde findet was er sucht, umso wahrscheinlicher ist seine Rückkehr zu dieser Plattform. Findet ein Nutzer nichts oder bekommt ungenaue Empfehlungen wird dieser nicht endlos suchen, sondern die Seite verlassen. Diese Systeme sind mittlerweile sehr weit verbreitet. Gängige Plattformen wie Amazon, Netflix, Spotify oder Google implementieren diese Systeme, um in gewisser Weise ein Alleinstellungsmerkmal gegenüber der Konkurrenz zu besitzen. Wenn man Spotify und seine wöchentliche Playlist als Vorbild nimmt, erhält jeder Nutzer basierend auf vergangen Interaktionen mit Songs neue Vorschläge und das jede Woche.

2.1.1 Arten von Bewertungen

Im Kontext von Recommender Systemen wird zwischen expliziten und impliziten Bewertungen unterschieden, die folgend kurz erläutert werden.
Explizit: Diese Bewertung wird von einem Benutzer direkt abgegeben. In Form von Daumen hoch/runter oder auf einer Skala. Der Informationsgehalt dieser Art hat seine Grenzen. Es lässt sich beispielweise keine Aussage ableiten, warum ein Benutzer eine Hose mit einer fünf bewertet. War die Lieferung langsam, war der Artikel beschädigt oder fällt die Passform unerwartet aus? Eine textuelle Bewertung wird daher meist hinzugefügt, was allerdings den Verbrauch an Ressourcen wie Serverkapazitäten und Datenspeicher erhöht.

Implizit: Hier wird numerisch versucht aus dem Verhalten eines Benutzers auf einer Plattform abzuleiten, inwieweit diesem ein Produkt zusagt. Dies kann beispielweise die Form eine Klick-Vektors annehmen, welcher die angeklickte Reihenfolge von Links auf der Webseite speichert und verarbeitbar macht oder durch die Betrachtungsdauer eines Abschnitts einer speziellen Webseite.

Zusätzlich zu den personalisierten Vorschlägen, die auf impliziten und expliziten Bewertungen, beziehungsweise Verhaltensweisen getroffen werden, gibt es auch die die nicht-personalisierten Empfehlungen. Folgend werden diese kurz erörtert.

Entpersonalisiert: Diese Art der Empfehlung nimmt beispielweise die Form von Top-10 Charts an oder Produkte, die häufig zusammengekauft werden, auch Assoziationsregeln genannt. Es gibt auch von Hand ausgewählte Produkte, die gesondert beworben werden sollen, beispielweise auf der Startseite eines Online-Shops. Es werden hierbei keine Rückschlüsse auf das Interesse eines Nutzers gezogen, sondern statistische Verfahren angewandt, um zum Beispiel den Durchschnittlich von dem am besten bewerteten Artikel einer Kategorie zu empfehlen. In Situationen, in denen wenig Nutzerdaten vorhanden sind, eignet sich diese Art der Empfehlung besonders.

2.1.2 Kollaboratives Filtern

Kollaboratives Filtern beschreibt, wie Inhalte einem Nutzer vorgeschlagen werden. Dabei wird davon ausgegangen das sich ähnlich verhaltende Benutzer sowohl in der Gegenwart als auch in der Zukunft symmetrisch verhalten. Interagiert ein Nutzer mit einem neuen vom System erstellten Empfehlung, das kann ein Kauf oder auch eine längere Betrachtung eines Artikels sein, so könnte dieser Gegenstand auch für die Nutzer in derselben Peripherie dieses Nutzers von Interesse sein und wird somit diesen Nutzern auch vorgeschlagen. Als Beispiel wird die wöchentliche persönliche Songempfehlung von Spotify aufgeführt. Hier besteht die Empfehlung aus einer gigantischen Matrix, in der jede Reihe einen Nutzer und jede Spalte ein Lied darstellt. Hat der Benutzer nun ein Lied mit einem Herz markiert bzw. gespeichert so wird für diesen Nutzer und Lied in dieser Zelle eine Eins eingetragen. Für alle übrigen Lieder, die von diesem Nutzer nicht gespeichert wurden, wird in diesem Fall eine Null in der entsprechenden Zelle eingetragen. Anzumerken ist das eine Null ebenso aussagekräftig ist wie eine Eins. Allerdings wird eine qualitative Empfehlung umso so schwieriger sollten Nullen oder Einsen übermäßig vertreten sein. Der Nutzer ist so zu beliebig beziehungsweise hat kein klares Profil. Ein großer Vorteil dieser Methode ist das inhärente Wissen, das sich dadurch generieren lässt. Die Notwendigkeit des spezifischen Wissens über Gemeinsamkeiten und Relationen unter den Objekten entfällt daher. Nachteile dieser Methode sind beispielweise neue Nutzer, die noch keinen Fußabdruck im System haben und es so schwierig wird ihnen automatisiert Dinge zu empfehlen. Dies nennt man auch Problem des Kaltstart. Weiter ist die Größe der Matrizen bei großen Online-Plattformen ein Problem, wie am Beispiel Spotify zu sehen. Da diese ein hohes Maß an Rechenkapazitäten bedingen durch die schiere Menge an verfügbaren Objekten beziehungsweise Spalten. Die Kosinus-Ähnlichkeit und der Korrelationskoeffizient nach Pearson sind hierbei häufig zum Einsatz kommende Methodiken. Folgend wird nur die Kosinus-Ähnlichkeit kurz erläutert. Kollabortives Filtern teilt sich in zwei Bereiche *model-based* und *memory-based*. (Falk 2019: 181 ff)

Die Kosinus-Ähnlichkeit wird definiert durch:

$$cos(\vec{x}, \vec{y}) = \frac{\sum_{i \in I_x y} r_{x,i} r_{y,i}}{\sqrt{\sum_{i \in I_x} r_{x,i}^2} \sqrt{\sum_{i \in I_y} r_{y,i}^2}} \tag{2.1}$$

Berechnet wird ein Grad an Ähnlichkeit zweier Vektoren, inwieweit sie in dieselbe Richtung im Raum zeigen. Der Wert kann dabei Werte aus Intervall [-1,1] annehmen, 1 bedeutet absolute Ähnlichkeit, 0 Unabhängigkeit voneinander und -1 eine diametrale Beziehung.

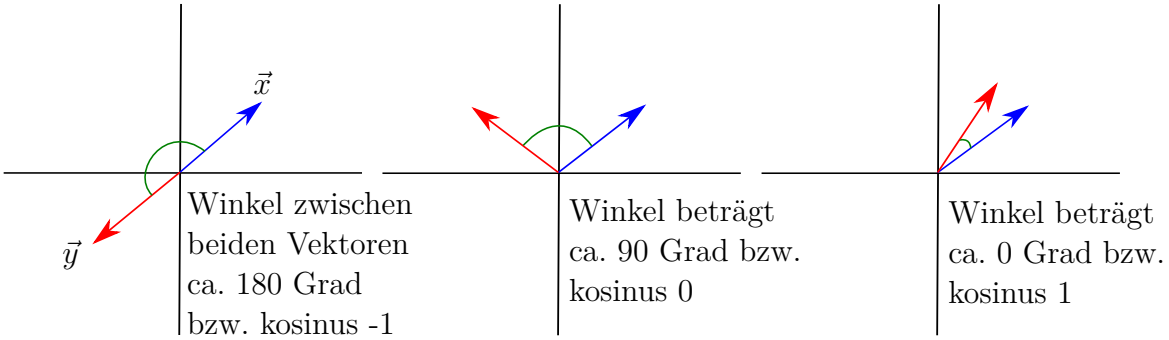

Abbildung 2.1: Kosinus-Ähnlichkeit: keine Ähnlichkeit (links), Neutralität (mittig), Ähnlichkeit (rechts)

Vorteile dieser Vorgehensweise sind die Erklärbarkeit der Resultate und die Simplizität der Entwicklung sowie die Benutzung derer. Großer Nachteil ist das Einbrechen der Erfolgsquote guter Empfehlungen, sofern Nutzer wenig mit Produkten auf der Plattform interagieren oder die Anzahl an verfügbaren Objekten mit denen interagiert werden kann, zu groß werden und dabei eine Matrix mit vielen leeren Zellen entsteht. Weiterer Nachteil ist die notwendige Neuberechnung bei jedem hinzukommenden Benutzer oder Produkt. Bei der model-based Vorgehensweise werden Algorithmen aus den Bereichen der Statistik sowie des maschinellen Lernens verwendet, um Vorhersagen über eine mögliche zukünftige Bewertung eines Nutzers zu machen. Darauf aufbauend wird häufig eine Rangfolge von möglichen Objekten erstellt, die dem Benutzer vorgeschlagen wird. Dabei ist die Rangfolge absteigend, beziehungsweise das Objekt mit der größten Wahrscheinlichkeit auf Platz Eins, das dem Geschmack des Nutzers entsprechen könnte. Zu dieser Vorgehensweise gehören unteranderen *Baye'sche Netze*, *k-means clustering* und *Singular Value Decomposition* (SVD). Vorteil ist hierbei das selbständige Erlernen von Eigenschaften, jedoch ist ein Nachteil die Erklärbarkeit der Resultate. Dagegen ist es mit model-based Methoden einfacher neue Objekte und User zum Pool hinzuzufügen, da Korrelationen selbständig abgeleitet werden. Anzumerken ist, dass sowohl eine Nutzer-Objekt Matrix oder auch eine Objekt-Objekt Matrix verarbeitet werden kann.

2.1.3 Inhaltsbasiertes Filtern

Inhaltsbasiertes Filtern beschreibt den Vorgang des Encodierens eines Produkts durch Metadaten wie auch textuelle Beschreibungen in nummerische Werte. Die Kunst bei dieser Methode ist es die sinnvollsten Eigenschaften, die dieses Produkt am besten beschreiben, zu finden und diese so zu transformieren damit ein Computer diese verarbeiten kann. Häufig werden dabei Metadaten des Produkts verwendet, um einen Vektor zu konstruieren der dieses Produkt repräsentiert. Anhand eines Songs wäre dies beispielweise die Anzahl an Bässen pro Sekunde oder auch Dauer der Höhen sowie Tiefen und viele andere Eigenschaften, die in diesen Vektor mit einfließen könnten. Dazu können aber auch Informationen textueller Natur kommen, wie etwa Genre, Künstlername, Album. Gefragt und wichtig hierbei ist das Domänenwissen über die Produkte und das Ziel, das der *Recommender* ultimativ verfolgen soll. In diesem Kontext ist die *Kosinus-Ähnlichkeit* zu nennen, die sehr häufig zum Einsatz kommt, wenn einem Nutzer Empfehlungen gemacht werden sollen, dieser jedoch bisher kaum mit der Online-Plattform interagiert hat. Hierbei wird innerhalb der Datenbank mit dem Nutzer-Vektor nach Produkten gesucht, die sich ähneln. Beispielweise auf Basis von Nutzereingaben bei der Anmeldung. Der Nutzer würde nur angeben müssen zu welchem Grad dieser sich für spezielle Produktkategorien interessiert. Vorteil dieses Verfahrens ist, das nur begrenzte Informationen über einen Nutzer vorliegen müssen da hierbei die Produkte vielmehr charakterisiert sind als der Nutzer. Das Problem des Kaltstarts entfällt also. Nachteil ist die Spezialisierung beziehungsweise das Fehlen der Übertragbarkeit auf andere Kategorien. Beispielweise könnte man innerhalb der Kategorie Film nicht auf Ergänzungen aus der Kategorie Nachrichten zurückgreifen. Da diese eventuell nicht die gleichen Sachen besprechen. (Falk 2019: 248 ff)

2.2 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) (Blei et al. 2003) ist ein generatives probabilistisches Modell, welches in den Bereich der Bayes‘schen Statistik fällt. Häufig angewendet in Fällen textueller Informationsgewinnung. Jedoch deuten die Autoren darauf hin, dass dieses Modell sich nicht nur auf textuelle Fragestellungen anwenden lässt, sondern jedes Szenario bearbeiten kann bei dem die einzelnen Entitäten als eine Wahrscheinlichkeitsverteilung von diversen Kernthemen zusammengefasst werden können. In der Fachliteratur werden diese Kernthemen *Topics* genannt. Dieser

Begriff wird ebenfalls in dieser Arbeit verwendet. Weiter wird LDA, in dieser Arbeit, ebenfalls im Kontext textuell vorliegender Information erklärt sowie verwendet.

Im Kontext von Zeitungsartikel wird folgend *LDA* erläutert, um eine Intuition für den Prozess zu fördern. Die Kernaussage dieses Modells lässt sich folglich als solches zusammenfassen. Jeder Zeitungsartikel lässt sich als Verteilung von *Topics* und diese *Topics* lassen sich wiederum als Verteilung von Wörtern zusammenfassen. Also ist davon auszugehen, dass die Anzahl bestimmter Wörter, die in zwei Zeitungsartikeln ähnlich oft vertreten sind, diese Artikel die gleichen *Topics* besprechen. Weiter ist die Anordnung oder Position der Wörter im weiteren Verlauf irrelevant. Wenn man Wörter vertauscht, bleiben die Kernaussagen eines Artikels dennoch ableitbar. Somit lässt sich jeder Artikel als ein *Bag-of-Words* Vektor auffassen, wodurch das *LDA* Modell wiederum diese als Datengrundlage erhält. Folgende Abbildung zeigt einen beispielhaften *Bag-of-Words* Vektor für Artikel.

Abbildung 2.2: Exemplarische *Bag-of-Words* Vektoren,Quelle: https://diveki.github.io/projects/wine/tfidf.html

Auf dieser Basis, sowie einer vorher festgelegten Anzahl an *Topics*, gruppiert dieses Modell die eingespeisten Zeitungsartikel in *Topics* anhand der enthaltenen Wörter. Häufig ist die Zahl der *Topics* A-priori eine Annahme, da sich die korrekte Anzahl an tatsächlichen *Topics* schwer erahnen lässt. Die Problematik der optimalen Anzahl von *Topics* wird im Kapitel Implementation näher beleuchtet. Folgende Abbildung zeigt *LDA* graphisch.

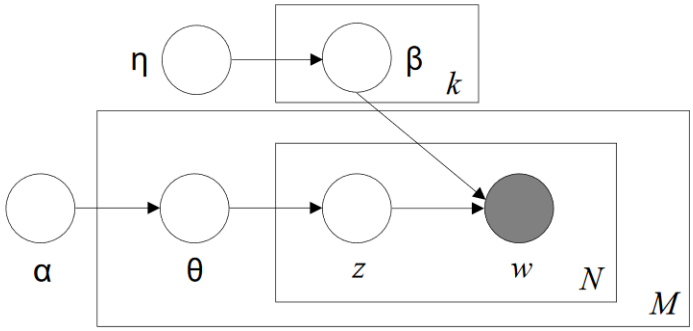

Abbildung 2.3: Graphische Darstellung *LDA* (Blei et al. 2003)

Es folgt eine Definition der Notationen:

- k = Anzahl Topics
- V = Größe des Vokabulars bzw. Anzahl der einzelnen Wörter
- M = Anzahl Entitäten bzw. Zeitungsartikel
- N = Anzahl Wörter innerhalb eines Zeitungsartikel
- w = Wort innerhalb eines Zeitungsartikel als *one hot encoded vector* mit der Dimension V
- $\mathbf{w}$ = der Zeitungsartikel bzw. Vektoren von w's mit N Wörtern
- $\mathcal{D}$ = Der Corpus bzw die gesamte Anzahl von Zeitungsartikeln

θ, β und z sind hingegen Verteilungen und keine Konstanten. θ ist eine Matrix wo θ(i,j) die Wahrscheinlichkeit für Artikel i festhält ob dieser die Wörter aus *Topic* j enthält und eine Größe von $(M \times k)$ aufweist. β(i,j) beherbergt die Wahrscheinlichkeit ob das i-te *Topic*, Wort j enthält, mit einer Größe von $(k \times V)$. Beide Verteilungen werden als Dirichlet Verteilung modelliert. z symbolisiert ein *Topic* als Verteilung von

Wörtern als Matrix mit Gestalt $(N \times k)$. α ist ebenfalls eine Matrix mit der Gestalt $(M \times k)$ und enthält für jeden Artikel eine *Topic* Verteilung. η hingegen, mit der Gestalt $(k \times V)$, repräsentiert den Parametervektor für jedes *Topic*, besser gesagt wie wahrscheinlich es ist, dass ein Wort zu diesem Topic gehört. Die Hyperparameter, α sowie η, können angepasst werden, um das *Topic* in gewissen Maß zu beeinflussen. Ein hoher Wert für α verteilt jeden Artikel auf viele Themen, somit wird der resultierende Vektor weniger nullen enthalten. Ein kleiner Wert für α verteilt nur wenige *Topics* auf einen Artikel. Der Vorteil bei hohen α-Werten ist, dass Artikel eine höhere Ähnlichkeit aufweisen werden und somit vergleichbarer erscheinen. Ein kleiner α-Wert wird die Artikel stark differenzierbar machen. Hier ist der Kontext wichtig beziehungsweise das Ziel, das man erreichen möchte. Ähnlich verhält es sich bei η. Ein hoher Wert für η führt dazu, dass *Topics* verwandter werden. Durch die Verteilung der Wahrscheinlichkeitsmasse der Zugehörigkeit von Worten zu den diversen *Topics* wird auf mehrere Wörter ebenmäßiger verteilt. Hierdurch wird eine vermehrte Überlappung zwischen *Topics* erzielt.

Da die genaue A-Posteriori-Verteilung für die Parameter β, θ und z für D bzw. die einzelnen Artikel des Corpus, nicht oder nur schwer lösbar ist, wird dies mithilfe der *Kullback-Leibler Divergenz*, iterativ approximiert bis zur Konvergenz. Folgender Term zeigt das Optimierungsproblem. γ, ϕ und λ repräsentieren variationale Parameter. Weiter verkörpern q und p die Verteilungen, auf welche die Kullback-Leibler Divergenz D angewendet wird.

$$\gamma^*, \phi^*, \lambda^* = argmin_{\gamma,\phi,\lambda} D(q(\theta, z, \beta \mid \gamma, \phi, \lambda) \mid\mid p(\theta, z, \beta \mid \mathcal{D}; \alpha, \eta)) \tag{2.2}$$

Durch diesen Lernprozess kann das Modell bisher ungesehene Artikel Topics zuteilen bzw. diesen einordnen. Die Ausgabe dieses Modells ist also eine Wahrscheinlichkeitsaussage zu welchen Topics und zu welchem Grad ein Artikel, laut Modell, eingeordnet wird. Es folgt eine beispielhafte Darstellung für den Aufbau eines Topics, anhand eines trainierten LDA Models. Die nummerischen Werte, Wörter Verteilung und Topic Verteilung, entsprechen Wahrscheinlichkeiten.

Pandemie: (0.34: „R-Zahl“, 0.25: „Robert-Koch-Institut“, 0.60: „neu-Ansteckungen“)
Corona: (0.25: „Bier“, 0.75: „Virus“)
Trump: (0.35: „ausfällig“, 0.50 „unberechenbar“, 0.15: „tweet“)
Fußball: (0.30:„Fifa“, 0.10:„Champions-League“, 0.30:„Bundes-Liga“, 0.30:„FC-Bayern“)

Würde man nun einem trainierten LDA Model einen Zeitungsartikel übergeben, immer im *Bag-of-Words* Format, der das Thema „Trumps Umgang mit dem Corona Virus“ behandelt, vermutlich würde die generierte Topic Verteilung folglich so aussehen.

Zeitungsartikel 1: (0.20 Pandemie, 0.40 Trump, 0.40 Corona, 0.0 Fußball)

Anzumerken ist das die Namen der *Topics*, wie oben beschrieben, nicht zur Ausgabe eines *LDA* Modells gehört, sondern von 1 bis N nummeriert sind. Die Hauptaussage der einzelnen *Topics* muss selbständig abgeleitet werden.

2.3 Künstliche neuronale Netze

In diesem Unterkapitel werden die Grundlagen zu künstlichen neuronalen Netzen, die Begrifflichkeiten und Algorithmen sowie einige Topologien erläutert.
Begründet durch die Ideen von Warren McCulloch und Walter Pitts im Jahr 1943, entstand der Bereich der künstlichen neuronalen Netze. In dieser Zeit postulierten die zwei Herren mit ihrer Idee der Nachahmung eines organischen Neurons, das Perzeptron, und der Vernetzung dieser künstlichen Neurone die künstlichen neuronalen Netze, die heute in verschiedenen Formen existieren. Durch die Analyse dieser Netze im Jahr 1969 durch Marvin Minsky und Seymour Papert und der daraus gewonnen Erkenntnis, dass diese Netze wichtige Probleme wie XOR-Operationen nicht lösen konnten, verfiel dieses Forschungsgebiet jedoch in einen Dornröschenschlaf. Erst 1985, durch die Lösung des Problems des Handlungsreisenden von John Hopfield mit Hilfe seines entwickelten Hopfield-Netz, sowie die Entwicklung der allgemeinen Anwendbarkeit der Fehlerrückführung im selben Jahr, welche Paul Werbos aber schon 1974 für seine Dissertation erdachte, belebte diesen Forschungsbereich wieder und erhielt dadurch wieder mehr Beachtung und Forschungsmittel. Mit Hilfe der Fehlerrückführung war es nun möglich nicht-linear separierbare Probleme mit mehrlagigen und vollständig verknüpften Perzeptronen zu lösen. Die endgültige Hochphase, wie man sie bis heute erlebt, erreichten KNNs mit der erfolgreichen Implementierung der Fehlerrückführung in tiefen vorwärts gerichteten KNNs, durch das Forscherteam um Yann LeCun an der New York University. Yann LeCun, Yoshua Bengio und Geoffrey Hinton werden zuweilen auch als *Godfathers of Deep Learning* tituliert und erhielten für ihre Errungenschaften im Bereich der künstlichen Intelligenz den Turing Award

(Goodfellow et al. 2016: 13-21).
Künstliche neuronale Netze bestehen immer aus einer Ein- sowie Ausgabeschicht und ein bis mehrere verdeckten Schichten dazwischen. In der Topologie der KNNs gibt es rekurrente Netze, die eine Rückkopplung bei Neuronen auf sich selbst einsetzen wodurch diese Netze in gewisser Art ein Gedächtnis erhalten und am häufigsten in Bereichen ihren Einsatz finden, wo es sich bei der Eingabe um eine Sequenz handelt wie zum Beispiel Text oder kontinuierliche Signale. Eine weitere Gruppe sind die vorwärts gerichteten Netze, die auf keine vergangenen Berechnungen zurückgreifen, die im Netz selbst stattfanden und somit die Eingabe durch das Netz zur Ausgabe schiebt. Diese Netze werden für Klassifikations- und Regressionsprobleme verwendet. Folgende Abbildung zeigt ein simples KNN.

Die Neuronen sind untereinander über Kanten verbunden, welche alle ihre eigenen Gewichte sowie Bias besitzen und für gewöhnlich mit randomisierten Zahlen initialisiert werden. Die Werte der Gewichte werden dann in der Trainingsphase iterativ verändert. Passiert ein Eingabeparameter diese Kanten, wird dieser Wert mit jedem dieser Kantengewichte und Kantenbias verrechnet und an die nächste Schicht weitergeben, worauf sich dieser Vorgang bis zur letzten Schicht wiederholt. Die Aktivierungsfunktion, die etwas später erläutert wird, entscheidet ob das annehmende Neuron, nach der Verrechnung von Gewicht, Bias und Eingangsdaten, *„feuert“* und somit die Berechnungen an die nächste Schicht weitergibt. Ein Netz lernt durch die Anpassung der Kantengewichte zwischen den Neuronen in der Trainingsphase bei kontinuierlicher Änderung dieser pro *Batch*. Hierbei wird, nach einer vorher festgelegten Anzahl, der gesamte Datensatz in kleine Teile von Datensätzen aufgeteilt. Diese Teilmengen werden *Batches* genannt. Diese *Batches* werden in der Trainingsphase fortlaufend in das KNN gespeist bis alle *Batches* das Netz einmal vollständig durchlaufen haben. Dies entspricht einer *Epoche*. Es können je nach Notwendigkeit mehrere *Epochen* durchlaufen werden, wodurch das Netz immer besser die Eigenschaften der Daten lernt und später im besten Fall vorher ungesehene Daten richtig klassifizieren kann. Auf die Trainingsphase folgt die Testphase. Hierbei wird überprüft, wie gut ein trainiertes Netz bisher ungesehene Daten klassifizieren kann. Ein Netz wurde beispielweise mit Bildern trainiert, die entweder Äpfel oder Birnen mit unterschiedlichen Farben, Formen und Position auf dem Bild zeigen. Nun werden vom Netz bisher ungesehene Bilder eingespeist und klassifiziert. Am Grad der richtigen

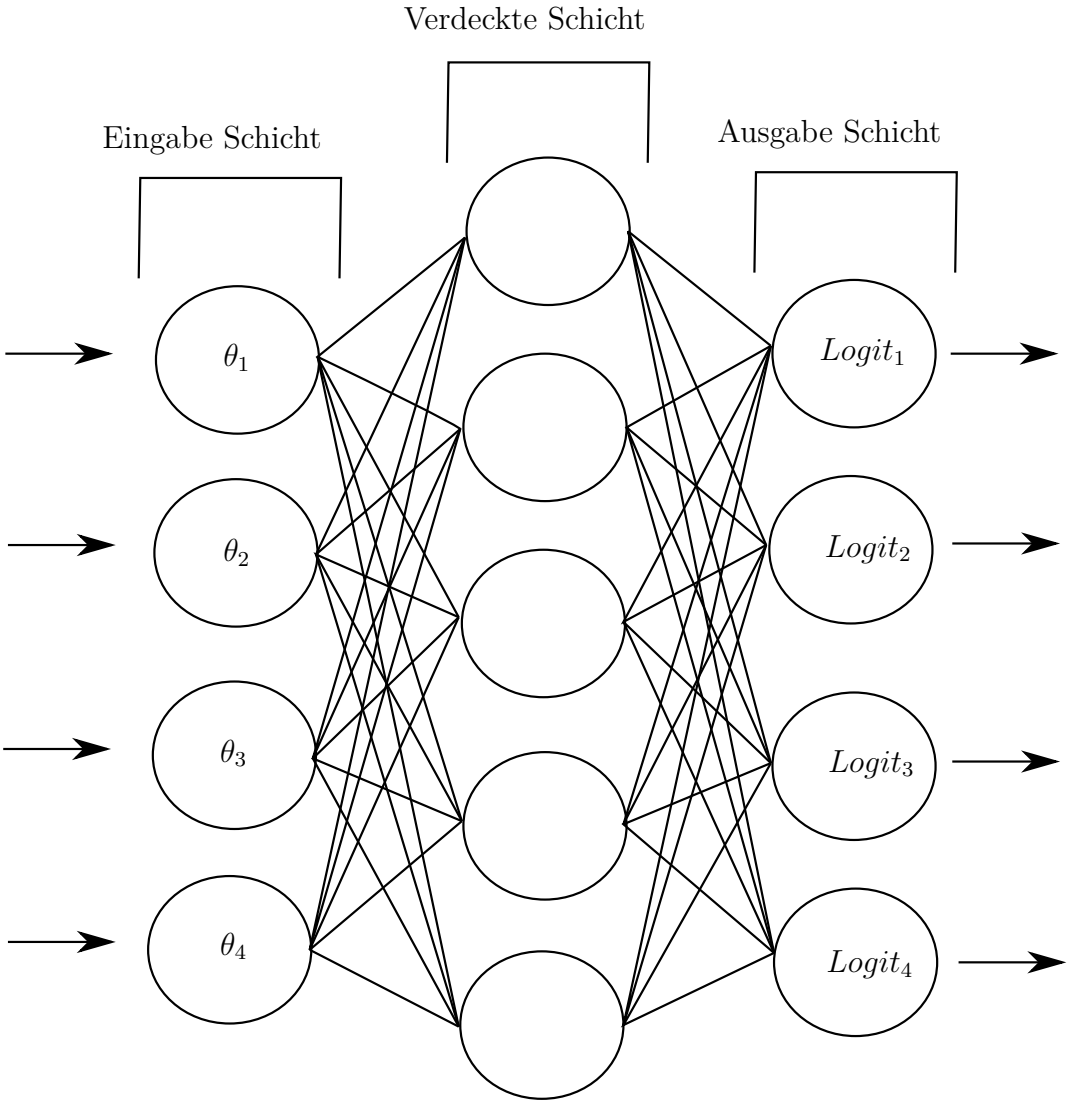

Abbildung 2.4: Beispielhafte Abbildung eines künstlichen neuronalen Netz

Klassifizierungen von Äpfeln oder Birnen lässt sich das Netz messen. Die Metrik ist hierbei die Genauigkeit der Vorhersage. Es existieren jedoch weitere Metriken mit denen man die Performanz eines KNNs messen kann und kommen auf den Kontext oder das gesteckte Ziel für das KNN an (Goodfellow et al. 2016: 168-181). Es kann beim Trainieren von KNNs eventuell zu folgenden zwei Problematiken kommen, welche kurz erläutert werden.

Überanpassung: macht sich bemerkbar wenn ein Modell die Trainingsdaten zu gut modelliert aber bei Testdaten versagt diese zu klassifizieren. Dies bedeutet, dass das Rauschen oder zufällige Schwankungen in den Trainingsdaten vom KNN aufgenommen und als Konzepte erlernt werden. Das Problem ist, dass diese Konzepte nicht auf neue Daten zutreffen und sich negativ auf die Genauigkeit von KNNs auswirken.

Unteranpassung bezieht sich auf ein KNN, das weder die Trainingsdaten noch Testdaten klassifizieren kann. Diese Problematik ist leicht zu erkennen da das KNN eine schlechte Leistung bei den Trainingsdaten aufweisen wird.

Die *Dropout-Schicht* ist eine spezielle Art unter den Schichten eines KNN. Hierbei wird anhand einer gewählten Rate zufällig Werte innerhalb der Eingabe auf Null gesetzt. Dies kann einer *Überanpassung* des KNN vorbeugen.

2.3.1 Kategorien des maschinellen Lernens

Im Folgenden werden kurz die beiden Unterkategorien des maschinellen Lernens erläutert, um eine bessere Eingrenzung zu vermitteln. Diese beiden sind jedoch nicht die einzigen Unterkategorien. Zu diesen gehören auch die Bereiche des bestärkenden Lernens sowie des stochastischen Lernens und das semi-überwachte Lernen.

Überwachtes Lernen: Bei dieser Kategorie ist die Ausgabe eines KNNs bereits bekannt und vordefiniert. Das Netz soll also selbständig lernen die Funktionen zu approximieren welche die Ausgabe am besten beschreibt. Die Beschreibung „überwacht“ rührt daher, dass genau beurteilt werden kann, ob das KNN richtig oder falsch in seiner Annahme ist zu welcher Ausgabeklasse die Eingabe gehört. Metrik ist hierbei die Vorhersagegenauigkeit.

Unüberwachtes Lernen: Hierbei ist die Zugehörigkeit unbekannt und das KNN soll selbständig latente Faktoren des Datensatzes erlernen. In gewisser Weise werden die Daten also eingruppiert, wodurch sich eine Aussage über die Struktur und Beschaffenheit der Daten treffen lässt, anhand der Zugehörigkeit zu einer Gruppe.

2.3.2 Aktivierungsfunktion

Im Hinblick auf den Lernprozess eines KNNs, sollen die Gewichte und die Bias einer jeden Schicht kontinuierlich in kleinen Schritten angepasst werden, gemäß der Akkuratheit eines KNNs. Ein Neuron hat jedoch lediglich die Zustände null und eins. Somit benötigt man eine stetige Funktion die Werte skaliert wodurch sich wiederum eine Aussage über die Stärke der berechneten Werte machen lässt. Die Gradienten der Ableitungen dieser Funktionen zeigen die Richtung des steilsten Abstieges. Jedes Neuron in einem KNN beherbergt eine Aktivierungsfunktion die darüber bestimmt, ob dieses Neuron feuert bzw. aktiviert wird und die errechneten Werte aus der vorherigen Schicht verarbeitet und in einem nächsten Schritt an die folgende Schicht weitergibt. Für gewöhnlich wird für jede Schicht eine Aktivierungsfunktion gewählt und nicht pro Neuron. Anzumerken ist das lediglich nichtlineare Funktionen gewählt werden sollten, da nur diese die komplexen Eigenschaften eines Datensatzes erlernen können und zusätzlich sich so Schichten stapeln lassen, bzw. DNNs ermöglicht.

Die Ausgabeschicht eines KNN benötigt jedoch, je nach Kontext und Problemstellung, eine spezielle Aktivierungsfunktion mit der die rohen Werte der Ausgabeschicht, fachsprachlich *Logits* genannt, verrechnet werden. Beispielweise wird für Multiklassen Probleme die *Softmax* Funktion verwendet, wodurch eine Wahrscheinlichkeitsverteilung im Intervall $[0, 1]$ für die N Ausgabeneuronen berechnet wird. Also bekommt die wahrscheinlichste Klasse den größten Anteil des Intervalls. Aber auch die *Sigmoid* Funktion ist eine gängige Aktivierungsfunktion, welche in Szenarien eingesetzt wird in denen entschieden werden muss ob eine Entität zu einer Klasse gehört oder nicht (Goodfellow et al. 2016: 182-187). Folgend werden die in dieser Arbeit zu Einsatz kommenden nicht linearen Aktivierungsfunktionen kurz vorgestellt. Die Auswirkungen auf das später implementierte KNN werden im Kapitel Evaluation eruiert.

Rectified Linear unit - ReLU:

$$f(x) = max(0, x) \tag{2.3}$$

ReLU transformiert eingehende positive Werte nicht und setzt negative Werte auf null. Hierdurch ressourcenschonend, da keine komplexe Berechnung durchzuführen ist. Diese Funktion ermöglicht zudem die Ausgabe von nullen nach der Aktivierung, welche im Kontext von Entitäten, die viele Nullen besitzen und diese wiederum eine wichtige Aussage über die Eigenschaft dieser Entität machen kann, ein wichtiges Merkmal ist (Goodfellow et al. 2016: 175).

Tanh:

$$f(x) = \frac{2}{1 + e^{-}2x} - 1 \tag{2.4}$$

Tanh hat den Vorteil der Bindung von Werten im Intervall $[-1, 1]$, also keine explodierenden Werte. Zudem helfen die steilen Gradienten, die wiederum eine stärkere Richtung für das Gradientenverfahren liefern. Großes Manko ist jedoch das *vanishing gradient* Problem. Dieses kann die Zeit für das Training eines KNNs erhöhen und eine verringerte Akkuratheit der Vorhersage verursachen.

Scaled Exponential Linear Unit - SELU:

$$f(x) = \lambda \cdot \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases} \tag{2.5}$$

Die Werte für α und λ sind Konstanten, die in dem originellen Artikel zu finden sind. *SELU* hat die Eigenschaft das KNN intern zu normalisieren. Die Werte von Gewichten, Bias und Aktivierungsfunktion werden also ein Mittel von null und eine Standardabweichung von eins haben, somit ist die Ausgabe eines KNNs ebenfalls normalisiert. Laut den Autoren, müssen, bei Verwendung dieser Funktion, die Gewichte des KNN mittels LeCun-Normal initialisiert werden. Wird Dropout in einem KNN verwendet, sollte Alpha-Dropout (Namensgebung in Tensorflow) benutzt werden (Klambauer et al. 2017).

2.3.3 Gradientenverfahren

Im englischen *Gradient Decent* genannt, versucht dieses Verfahren ein globales Minimum einer reellwertigen differenzierbaren Funktion zu finden. Es wird hierbei stetig und iterativ bis zur Konvergenz den am steilsten absteigenden Gradienten der Ableitung einer Funktion nach entlang gegangen. Auf diesem Pfad soll, im besten Fall schnellstmöglich, das globale Minimum einer gegebenen Funktion gefunden werden, wodurch die Aufgabe erfüllt wäre.

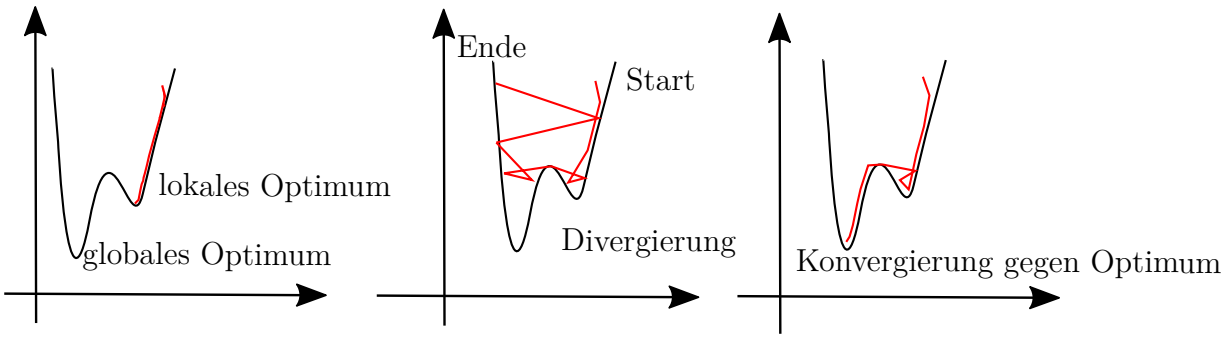

Abbildung 2.5: Exemplarische Darstellung des Gradientenverfahren

Auf Abbildung 2.5 ist der Vorgang zu sehen wie der Algorithmus Koordinate für Koordinate der Funktion abtastet und, nach dem steilsten Abstieg gehend, seine nächste Station wählt. Zu beachten ist hierbei der gewählte Wert für α, welches auch *Lernrate* genannt wird und die Schrittgröße des Algorithmus kontrolliert. Ein zu großer Wert für α lässt den Algorithmus eventuell nicht das Minimum finden beziehungsweise würden die Koordinaten auf dem Graph oszillieren und dadurch das Verfahren divergieren. Ein zu kleiner Wert führt unter Umständen dazu, dass der Algorithmus nicht gegen das optimale bzw. globale Minimum konvergiert, sondern in einem lokalen Minimum verweilt. Dies zeigt folgende Abbildung für die zwei Fälle mit einem nicht auf die Datenlage abgestimmten Wert für α. In Fällen höherer dimensionaler Funktionen, werden multiple lokale Optima existieren aber gesucht ist stehts das globale Minimum. Die Überprüfung ob während des Trainings das globale Minimum gefunden wurde, lässt sich durch mehrere Trainingsdurchläufe testen. Verbessern sich die gesuchten Metriken nach mehreren Testläufen nicht, so lässt dies die Annahme zu, dass das globale Optimum gefunden wurde.

Zusätzlich zu dem „normalen“ Gradientenverfahren wurden weitere Variationen dieses Algorithmus entwickelt, welche bei unterschiedlichen Problemstellungen bessere oder auch schlechtere Ergebnisse liefern im Hinblick auf die Schnelligkeit des Konvergierens. Der Unterschied zwischen dem stochastischen und „normalen“ Gradientenverfahren besteht darin, dass das stochastische Verfahren pro Iteration *Mini-Batches* verwendet. Damit werden die einzelnen Gewichte und Bias des KNN öfter aktualisiert (Goodfellow et al. 2016: 151, 171). Folgend wird ein Algorithmus vorgestellt, welcher eine Abwandlung des stochastischen Gradientenverfahren ist und in dieser Arbeit Verwendung findet.

Adaptive moment estimation – Adam: (Kingma et al. 2015) Dieser Algorithmus passt, im Gegensatz zu dem stochastischen Gradientenverfahren, fortlaufend und in Echtzeit Lernrate α an. Zudem bietet er für jeden einzelnen Parameter einen eigenen Lernkoeffizienten. Zusätzlich zu einem Speichern von vorhergehenden Gradienten innerhalb einer Iteration, welche fortlaufend an Gewicht verlieren, speichert Adam ein exponentiell verfallendes Mittel vergangener Gradienten. m_t und v_t sind Schätzungen des ersten Momentums, des Mittelwerts, und des zweiten Momentums, die unzentrierte Varianz, der Gradienten, daher der Name der Methode. Durch die Initialisierung von m_t und v_t als Nullvektoren, haben diese einen Bias Richtung Null zu Beginn des Prozesses. Die Autoren verhindern dies durch die korrigierte Berechnung, anhand des Bias, der ersten Momentum. W_t ist das Gewicht zur Iteration t. L_t verkörpert die Verlustfunktion. ϵ verhindert eine Division durch Null, welche einer kleinen natürlichen Zahl entspricht und β_1 sowie β_2 dienen der Skalierung des Verfalls vorhergehender Gradienten. Folgende Abbildung zeigt den kompletten Algorithmus.

2.3.4 Fehlerrückführung

Wichtiges Kernstück eines jeden vorwärts gerichteten KNNs ist die Fehlerrückführung, im englischen *Backpropagation* genannt. Das Rückführen des Ausmaßes eines Vorhersagefehlers findet statt, nachdem die Informationen vorwärts durch das Netz geflossen sind. Daher begründet sich auch der Begriff *vorwärts gerichtetes KNN*. Diese Methodik berechnet den Gradient des Fehlers an der Ausgabeschicht eines Netzes und passt, ausgehend von der Ausgabeschicht, hin zu der Eingabeschicht, die Gewichte der Kanten in einem KNN adaptiv an. Der Gradient einer Schicht wird zum Teil für die Berechnung des Gradienten der vorhergehenden Schicht verwendet. Somit

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Abbildung 2.6: Adaptive moment estimation - Adam (Kingma et al. 2015)

fließt der Gradient des Fehlers einer Schicht durch das Netz zurück. Dieser Algorithmus legt nicht fest zu welchem Zweck die berechneten Gradienten genutzt werden, dies legt die Optimierungsfunktion bzw. das Gradientenverfahren fest. Ist der Fehler entsprechend groß, werden die Gewichte dementsprechend stark angepasst. Gleiches gilt für den Fall eines niedrigen Vorhersagefehlers, dann werden die Gewichte weniger stark angepasst. Folgend wird der generelle Ablauf anhand einer Klassifikation dargestellt, im Kontext des überwachten Lernens in einem vorwärts gerichteten KNN (Goodfellow et al. 2016: 204-221).

$\vec{x_d}$ symbolisiert die Eingabedaten, y_d die zugehörige Klasse, $\hat{y_d}$ die von dem KNN prognostizierte Klasse, j ein Neuron, k eine Schicht des KNN, o der berechnete Wert durch die Aktivierungsfunktion und E der berechnete Fehler durch die ausgewählte Verlustfunktion.

1. **Berechnung der Vorwärtsphase:** Berechne für jedes Paar (x_d, y_d) die Vorwärtsphase und speichere $\hat{y}$ und o_j^k von Schicht Eins, die Eingabeschicht, bis Schicht m, die Ausgabeschicht.
2. **Berechnung der Rückwärtsphase:** Berechne für jedes Paar (x_d, y_d), ausgehend von Schicht m hin zu Schicht 1, die Rückwärtsphase und speichere für jede Kombination die partiellen Ableitung von E_d und $w_i j^k$ via zweimaliger

Anwendung der Ketten-Regel und führe dies

3. Kombiniere die durchschnittlichen Gradienten eines Paares mit den Gradienten anderer Paare aus dem *Batch* und die individuellen durchschnittlichen Gradienten, um die vollständigen Gradienten dieses *Batches* zu erhalten.
4. 4. Aktualisierung der Gewichte gemäß der gewählten Lernrate α, der vollständigen Gradienten aus Schritt drei und des gewählten Gradientenverfahren.

2.3.5 Variational Autoencoder

Der *Variational Autoencoder* (Kingma et al. 2013) ist eine abgewandelte Form eines künstlichen neuronalen Netz, welches Autoencoder genannt wird. Beide haben jedoch nur die symmetrische trichterförmige Anordnung der Schichten gemeinsam. Bei einem Variational Autoencoder handelt es sich zudem um ein probabilistisches Netz, wohingegen ein Autoencoder von deterministischer Natur ist. Dies ergibt sich durch die Art wie Autoencoder Eingabedaten in dem Flaschenhals, welcher in der Fachliteratur *latenter Raum* und *latent space* im englischen tituliert wird, komprimiert werden. Generell verfügen VAEs über zwei neuronale Netze, Encoder und Decoder, die durch einen sogenannten Flaschenhals verbunden sind und verkettet den VAE ergeben. Siehe Abbildung **??** für eine schematische Darstellung. Oftmals werden, in einschlägiger Literatur, Encoder, Inferenz Netz (englisch: inference net) und Decoder, Probenahme Netz (engisch: sampling net) genannt. Durch seine Fähigkeit, selbständig die wichtigsten Eigenschaften von Eingabedaten zu erlernen und ultimativ dadurch zu begreifen was die Struktur der Daten ausmacht, ist der VAE in der Lage neue Daten auf Basis des erlernten zu generieren. Mit dieser Selbständigkeit lässt sich ein VAE hervorragend im Bereich des unüberwachten Lernens einsetzen

Der Encoder besteht aus einer Eingabeschicht und ein bis mehrere verdeckten Schichten. Die verdeckten Schichten sind in ihrer Mächtigkeit an Knoten kleiner als die vorherige Schicht, wodurch der VAE selbständig die bestmögliche Komprimierung der Eingabedaten selbständig erlernen muss. Mehr dazu im Unterkapitel Verlustfunktion eines Variational Autoencoders. Der Flaschenhals ist die Verbindung zwischen Encoder und Decoder. In diesem Raum befinden sich die encodierten Datenvektoren als komprimierte Datenpunkte. Bei der Encodierung, die Ähnlichkeiten zu einer Methode namens „Principal component analysis“(PCA) aufweist, versucht das Netz eine möglichst verlustfreie Komprimierung des Datenvektors durchzuführen. Essen-

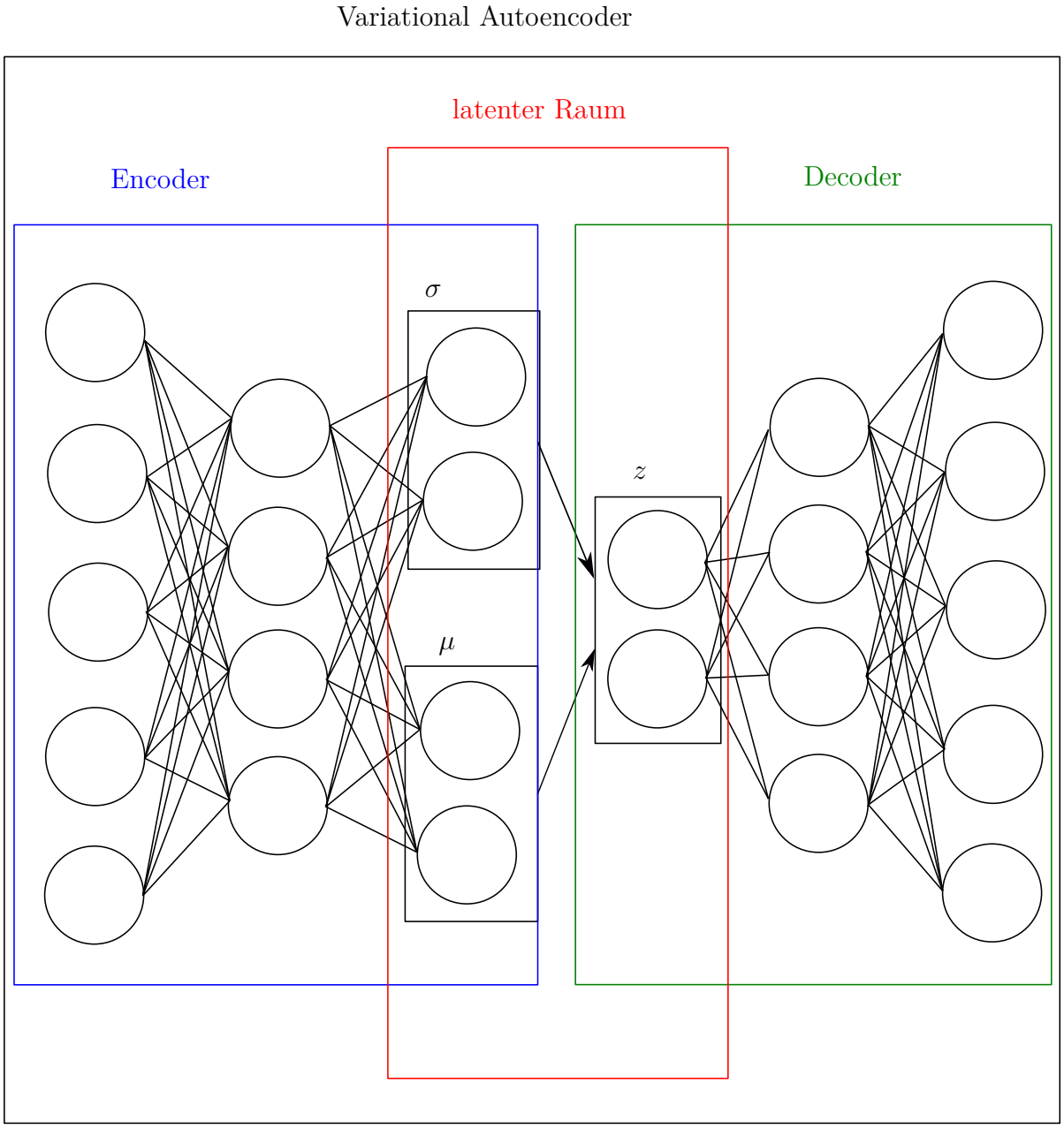

Abbildung 2.7: Schematische Darstellung eines Variational Autoencoder

zielle Dimensionen, die dieses Objekt eindeutig beschreiben, werden beibehalten und unwichtige verworfen, wodurch der Datenvektor, in den linearen Raum bei einem Autoencoder, und bei einem VAE als Verteilung projiziert wird. Wichtiger Unterschied zwischen VAE und Autoencoder ist die Tatsache der Encodierung des eingefügten Datenvektors nicht als einzelner Datenpunkt, sondern als eine Gaußsche Normalverteilung in den latenten Raum. Bewerkstelligt wird dies durch die Berechnung des Durchschnitts und Standardabweichung des eingefügten Datenvektors durch den Encoder, kurz vor dem latenten Raum. Daraufhin wird mittels des *reparameterization trick* die latente Repräsentation der Eingabedaten berechnet.

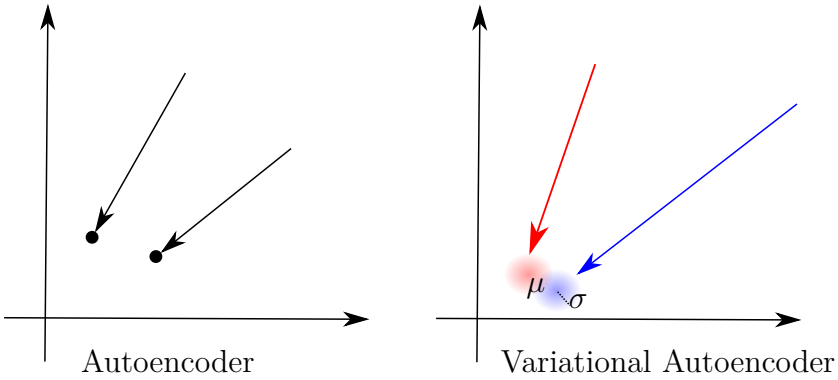

Abbildung 2.8: Darstellung der Komprimierung duch Autoencoder und Variational Autoencoder

Aufgabe des Decoders ist es aus den komprimierten Datenvektoren, die latenten Repräsentationen, eine bestmögliche Rekonstruktion der Eingabedaten durchzuführen. Dabei findet eine Probenahme der Datenpunkte aus dem latenten Raum statt, diese werden rekonstruiert und mit dem Eingabevektor verglichen. Wie bei neuronalen Netzen üblich, wird die Verlustfunktion iterativ minimiert und das Ausmaß der Trainingsfehlers mit Hilfe der Fehlerrückführung durch das Netz zurückgeschickt. Die Verlustfunktion eines VAE wird im Unterkapitel *Fehlerrückführung bei einem Variational Autoencoder* näher beleuchtet. Zielsetzung eines VAE ist eine möglichst nahe Wiederherstellung der Eingabedaten, nach ihrer Komprimierung.

Anwendung findet dieses Netz häufig im Bereich der Bildverarbeitung. Als Beispiel ist die Komprimierung eines hochauflösenden Bildes zu nennen. Hierbei wird das Bild zunächst mit Hilfe eines trainierten Encoders komprimiert und darauffolgend

von Server zum Benutzer geschickt woraufhin ein Decoder dieses Bild im Browser rekonstruiert. Wodurch die Menge an verwendeten Daten reduziert werden kann, die übertragen werden müssen. Für das menschliche Auge ist kaum ein Unterschied in der Qualität des Bildes, nach der Rekonstruktion, erkennbar. Die komprimierten Datenvektoren können auch als Eingabe für andere KNNs verwendet werden die beispielweise eine Klassifizierung durchführen wollen, dabei aber so sparsam wie möglich mit Hardwareressourcen umgehen sollten (Shafkat 2018), (Kingma et al. 2019).

2.3.5.1 Verlustfunktion eines Variational Autoencoder

Die Verlustfunktion eines Variational Autoencoders besteht aus zwei Termen, zum einen ein Rekonstruktionsterm und zum anderen ein regulatorischer Term. Ersterer berechnet den Fehler zwischen Ein- und Ausgabe. Zweiter ist für die Regularisierung des latenten Raums zuständig. Bei dem zweiten Ausdruck handelt es sich oftmals um die *Kullback-Leibler Divergenz*. Diese misst wie weit zwei Verteilungen voneinander divergieren. Eine Null als Ergebnis würde absolute Übereinstimmung bedeuten. Hierbei ist die Regulierung des latenten Raums besonders wichtig, denn ohne sie würden encodierte Objekte zu weit voneinander im Raum projiziert werden und so eine Generalisierung verhindern beziehungsweise wäre eine sanfte Interpolation zwischen encodierten Objekten nicht möglich. Um sicher zu stellen das Daten, die eine Ähnlichkeit aufweisen, in dem latenten Raum nah beieinander sind, muss das Mittel sowie die diagonale Kovarianzmatrix eines Eingabevektors einreguliert werden. So wird das Resultat des Encodierungsnetzes, die Verteilung, dazu gebracht einen Durchschnitt nahe Null und eine Kovarianz Matrix nahe der Identität zu besitzen. Was wiederum einer Standardnormalverteilung entspricht. Im Grunde erhält man dadurch „Datenwolken“, die sich um den Nullpunkt befinden, sich bei einer Ähnlichkeit auch überlappen können und zu einem gewissen Grad die Zuordnung eines Objekts zwischen unterschiedlichen Wolken zulässt. Die *Kullback-Leibler Divergenz* lässt sich für einen VAE wie folgt berechnen.

$$D_{KL} = \frac{1}{2} \cdot \sum_{i=1}^{n} \sigma_i^2 + \mu_i^2 - log(\sigma_i) - 1 \tag{2.6}$$

Berechnet wird somit die Summe über aller Komponenten der latenten Eingabedaten mit $X_i \sim \mathcal{N}(\mu, \sigma)$.

Der Rekonstruktionsterm hingegen wird vom Kontext des Problems bestimmt. Generell lässt sich jedoch sagen, dass ein VAE trainiert in dem versucht wird den sogenanntenevidence lower bound, abgekürzt ELBO, zu maximieren. Zusammen mit der *Kullback-Leibler Divergenz* bilden diese die Verlustfunktion eines VAE. Mit den *variational Parameter* θ und ϕ sowie x als Eingabe und z als komprimierte Eingabe , lässt sich die Verlustfunktion allgemein definieren durch:

$$\mathcal{L}_{\theta,\phi}(x) = logp_\theta(x) - D_{KL}(q_\phi(z \mid x) \mid\mid p_\theta(z \mid x)) \tag{2.7}$$

Hiermit werden die *variational Parameter* approximativ die marginale *Likelihood* maximieren, was ein besseres generatives Netz erzeugt. Sowie die *Kullback-Leibler Divergenz* minimieren, wodurch sich $q_\phi(z|x)$ immer näher an $p_\theta(x|z)$ angleicht (Kingma et al. 2019: 15-20).

2.3.5.2 Fehlerrückführung bei einem Variational Autoencoder

Folgend wird der Ablauf der Fehlerrückführung bei einem VAE erläutert, welcher sich durch die probabilistische Natur eines VAE doch deutlich von einem diskreten KNN unterscheidet.

Im Hinblick auf die im Unterkapitel VAE erläuterten Gesetzmäßigkeiten dieses Netzes, die Encodierung der Eingabedaten als Verteilung und Decodierung via Probenahme von dieser Verteilung, fällt unter Umständen auf, dass es sich bei dem latenten Raum zwischen Encoder und Decoder, um randomisierte Werte handeln. Bei der Fehlerrückführung bedarf es allerdings deterministischer Werte, um zu funktionieren. Um dennoch eine Fehlerrückführung auf das gesamte Netz, sprich Encoder und Decoder, zu ermöglichen, wird der *reparameterization trick* angewandt.

Wie in der Abbildung zu sehen, wird durch das Hinzufügen von ϵ an den sample Term, z deterministisch. Vorher waren die Komponenten dies nicht. Die Werte für ϵ werden dabei randomisiert von einer Normalverteilung bezogen. Somit wird z deterministisch, behält durch ϵ allerdings eine gewisse Stochastizität. Folgender Term zeigt die Berechnung von z:

$$\epsilon \sim \mathcal{N}(0, 1)$$

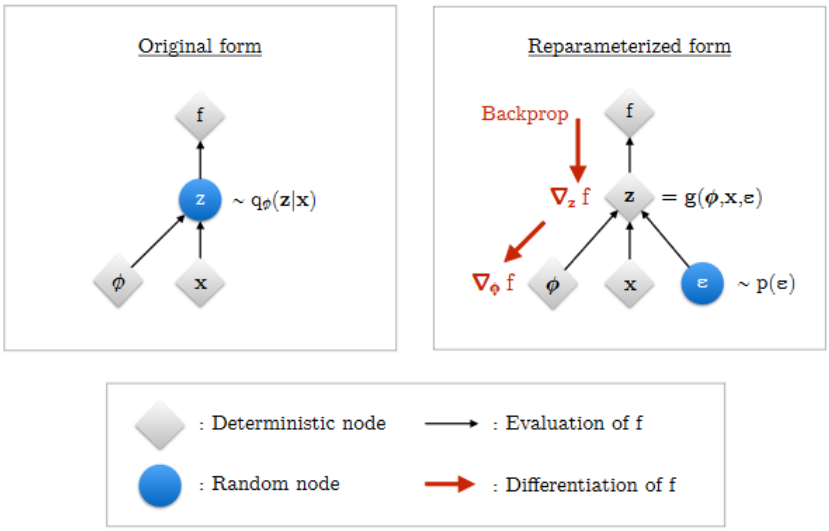

Abbildung 2.9: Fehlerrückführung via *reparameterization trick* bei einem VAE (Kingma et al. 2019: 22)

$$z = \mu + \sigma \odot \epsilon \tag{2.8}$$

Dadurch kann die Fehlerrückführung durch den VAE fließen. Anzumerken ist, das ϵ immer noch nicht deterministisch ist und somit darauf keine Fehlerrückführung angewendet werden kann, jedoch sind nur die Werte Durchschnitt und Standartabweichung der Eingabedaten für den eigentlichen Prozess von Interesse.

2.3.5.3 Variational Autoencoder Variationen

Multinomial-VAE: Diese Variante des VAE (Liang et al. 2018) beschäftig sich mit der Frage, unter der Anpassung der Rekonstruktionsfunktion, wie performant und anwendbar im Bereich des Kollaborativen Filterns ein VAE sein kann. Basierend auf einem Datensatz mit impliziten Bewertungen und dem Einsatz einer *Multinomial Likelihood* als Rekonstruktionsterm bzw. das Anwenden der *Softmax* auf die Logits des Netzes multipliziert mit dem Eingabevektor. Zusätzlich implementiert dieser Artikel ebenfalls den Koeffizienten des *beta-VAE* und erzielt damit eine signifikante Performanz in den erörterten Tests. Dieses wird ebenfalls später implementiert. Diese Variation, *Multinomial-VAE* in Verbindung mit *beta-VAE*, wird in dieser Arbeit implementiert und auf gewisse Szenarien angewandt, auf die im Kapitel Konzept eingegangen werden.

beta-VAE: In dem korrespondierenden Artikel (Higgins et al. 2016) wird beschrieben welche Auswirkung das Anbringen eines Koeffizienten an die Kullback-Leibler Divergenz, im Kontext eines VAE, hat. Hierdurch werden die Funktionen innerhalb des Fehlerterms unterschiedlich gewichtet. So soll β größer eins bis N einem VAE ermöglichen spezifische Eigenschaften eines Datensatzes zu erlernen, die Rekonstruktion aber unschärfer wird. Die Autoren sprechen von einer „Entwirrung“ von Eigenschaften (engl. Disentanglement). Wohingegen bei β kleiner eins ein größeres Augenmerk auf die Rekonstruktionsgenauigkeit gelegt wird. Später in dieser Arbeit wird dieser Koeffizient verwendet werden und ebenfalls, im Kontext des verwendeten Datensatz, evaluiert

3 Konzept

Dieses Kapitel erläutert die Zielsetzung und Fragestellungen dieser Arbeit. Die im Kapitel Grundlagen erörterten Algorithmen und Methodiken werden im folgenden Kapitel konkret implementiert und in dem darauffolgenden Kapitel anhand der Fragestellung evaluiert, welche in diesem Kapitel erörtert werden. Der *Variational Autoencoder* ist das Mittel zur Lösung der gegebenen Fragestellungen. Im Fokus steht der Bereich des inhaltsbasierten Filterns. Weiter werden nur einzelne Änderungen an den Algorithmen pro Versuch vorgenommen, um eine generelle Vergleichbarkeit zu ermöglichen, daraufhin werden nur die vielversprechendsten Modelle vorgestellt und weiterverwendet. In den beigefügten Dateien lassen sich weitere Graphiken zu den aussortierten Modellen finden. Um das Verhalten zu testen wird eine Bewertungsskala eingeführt. Die Bewertungen bewegen sich zwischen eins und fünf, zeitgleich bleibt die null erhalten, wo noch keine Bewertung von dem Nutzer abgegeben wurde. Anschließend werden die Nutzer-Vektoren immer normiert, um diese vergleichbar mit den Topic-Vektoren der Produkte zu machen.
Der Datensatz, welcher für das Trainieren des VAE verwendet wird, soll durch ein LDA Modell generiert werden. Bei dem eigentlichen Datensatz handelt es sich um Amazon Daten der Kategorie Videogames (Amazon Dataset 2018). Die einzelnen Daten enthalten textuelle Informationen wie Titel, Beschreibung und Eigenschaften. Bei den Eigenschaften handelt es sich um Informationen wie bspw. PC, Konsole, Accessoire, Ps4, also Kategorien. Diese Rohdaten werden nach einer Bereinigung für das Trainieren des LDA Models verwendet. Hierdurch ist das Prozedere um LDA als Zwischenschritt im Bereich der Datenvorverarbeitung zu begreifen und wird daher bereits im folgenden Kapitel evaluiert. Dort werden die Szenarien besprochen, welche mit Hilfe eines speziell für diese Fälle trainierte VAEs, mit Hinblick auf *beta-VAE* und *Multinomial-VAE*, auf ihren Nutzen evaluiert werden und ob diese einen Mehrwert generieren.

3.1 Szenario - 1: Kombination von Nutzer und Gegenstandvektor

Die Idee hierbei ist das ein Nutzer die Webseite eines Produkts betritt und darauf der Topic-Vektor des Produkts mit dem Topic-Vektor des Nutzers addiert werden soll. Der Nutzer würde auf einer Skala von null bis fünf auswählen, welche *Topics* zu welchem Grad für ihn infrage kommen würden. Durch diese Befragung lässt sich ein Topic-Vektor konstruieren. Der Nutzer-Vektor wird normalisiert daraufhin werden beide durch den VAE encodiert. Dann wird die Differenz $\vec{diff} = \vec{nutzer} - \vec{produkt}$ der encodierten Vektoren mit dem encodierten Nutzer-Vektor addiert. Daraufhin wird dieser transformierte Nutzer-Vektor von dem VAE decodiert. Zudem kann ein trainierter *Multinomial-VAE* auf diesen Vektor angewandt werden. Hierbei wird das Durchschnitt-Vektor des encodierten Vektors dem Decoder übergeben und auf die Logits die *Softmax* Funktion angewandt wodurch die nullen des Vektors mit entsprechenden Wahrscheinlichkeiten für die *Topics*, gefüllt werden sollen. Mit diesem Vektor wird in dem Datensatz mittels Kosinus-Ähnlichkeit und einem Schwellwert nach Produkten gesucht, die eine Ähnlichkeit anhand ihres *Topic-Vektor* aufweisen. Das Resultat dieser Abfolge könnte dann unter der Beschreibung des Produkts als ähnliche Produkte aufgelistet werden. In gewisser Weise wird hierbei das inhaltsbasierte und kollaborative Filtern kombiniert. Es wäre ebenfalls möglich nur mittels der Addition des Nutzer-Vektor und Differenz-Vektor nach ähnlichen Produkten zu suchen. Diese beiden Möglichkeiten sollen evaluiert werden.

3.2 Szenario - 2: Verfall von Nutzer Bewertungen

Hierbei stellt sich die Frage wie ein VAE auf verfallende Bewertungen reagiert. Die Idee hierbei ist, die bereits abgegebenen Bewertungen beizubehalten und einen zeitlichen Verfall dieser anzuwenden. Der zeitliche Verfall einer Bewertung soll, die sich stetig verändernden Präferenzen eines Benutzers darstellen und somit den Einfluss vergangener Interaktion auf aktuelle Produktempfehlungen begrenzen. Berechnet wird der Verfallsfaktor durch folgende Funktion.

$$f(x) = \sqrt[d]{\frac{Bewertung}{Heute - TagderletztenInteraktion}} \tag{3.1}$$

Hierbei stellt d einen zufällig ausgewällten Faktor dar, der nach Belieben geändert werden kann. Jedoch sollte immer eine positive gerade Zahl gewählt werden, um negative Werte zu vermeiden. Je kleiner der gewählte Wert, desto schneller verfällt die Bewertung und verliert somit an Bedeutung. Wie bereits in dem vorangegangenen Szenario, werden die Nutzer-Vektoren im Kontext der Topic-Vektoren behandelt. Darauf werden die entsprechenden Produkte in dem Datensatz mittels Kosinus-Ähnlichkeit sowie einem Schwellwert gesucht und zurückgegeben.

3.3 Szenario - 3: Nutzervektor mit vielen nullen

Dieses Szenario behandelt die Thematik des Kaltstarts im Kontext von Recommender Systemen. Das Problem des Kaltstarts beschreibt den Umstand, wenn ein Nutzer sich neu auf einer Plattform anmeldet und es aufgrund dessen keine Historie über diesen Nutzer von Interaktionen mit Produkten gibt. Somit ist es schwierig personalisierte Empfehlungen zu machen. Dieses Szenario behandelt die Fragestellung, wenn ein Nutzer lediglich ein oder zwei Topics ausgewählt und ein *Topic* auch nur mit beispielweise ein oder zwei auf der eingeführten Skala bewertet hat. Hierbei soll lediglich versucht werden herauszufinden wie ein VAE diesen Nutzer-Vektor rekonstruiert und daraufhin untersucht ob die ausgebenden Gegenstände Sinn ergeben.

4 Implementation

In diesem Kapitel werden die technischen Mittel, die Vorgehensweise bezüglich der Datenverarbeitung sowie das LDA Model implementiert sowie evaluiert. Weiter wird auch kurz auf die angepasste Architektur des verwendeten VAE eingegangen.

4.1 Technische Hilfsmittel der Umsetzung

Python (Version 3.7.7): 1991 hatte diese interpretierte und multi-paradigmen Programmiersprache ihre erste Veröffentlichung. Entwickelt wurde sie von dem niederländischen Informatiker Guido van Rossum. Sie unterstützt das objektorientierte sowie das funktionale Programmierparadigma und zeichnet sich durch ihre Einfachheit und Reduzierung von Semantiken aus sowie eine sichtbare Nähe zu Pseudo-Code. Was diese sehr leicht erlernbar macht. Python wurde 2020 durch Python3 ersetzt, wobei Python der letzte Patch mit der Nummer 2.7.18 zuteilwurde und seitdem nicht mehr weiterentwickelt oder unterstützt wird (Python 2020).

Tensorflow (Version 2.2): Entstanden aus dem von Google Brain entwickelten Framework DistBelief und mittlerweile von Google selbst erweitert und gewartet, ist heute bereits in Version 2.2 zu haben. Version 1.0.0 erschien im Jahr 2017. Hinzu kommt die Möglichkeit Berechnungen von der GPU, anstatt der CPU durchführen zu lassen, um eine gesteigerte Geschwindigkeit der Berechnungen zu erreichen. Tensorflow bietet für Python und andere Programmiersprachen eine API zur einfachen Implementierung von tiefen neuronalen Netzen, in ihren unterschiedlichsten Variationen, sowie einigen Algorithmen und Funktionen für diese Netze. Zusätzlich kümmert sich Tensorflow ohne Zutun des Entwicklers um die Fehlerrückführung. Entwickelt wird es in der Programmiersprache C++, da diese eine höhere Geschwindigkeit als Python aufweist. Tensorflow bietet Verknüpfungen zu Keras oder auch Tensorflow-Propability, welche auf Tensorflow Core aufbauen und weiter abstrahieren. Keras

zum Beispiel macht es möglich in wenigen Zeilen Code, ein neuronales Netz aus mehreren Schichten zusammen zu stellen. Großer Vorteil von Tensorflow gegenüber seinem direkten Konkurrenten PyTorch, das von Facebook entwickelt wird, der relative Marktanteil, beziehungsweise werden viele Prototypen aus der Forschung zuerst mit Tensorflow umgesetzt, was es wiederum einfacher macht aktuelle Forschung nachvollziehen zu können (Tensorflow 2020).

Scikit-learn (Version 0.23.1): Gestartet bei Google als Summer Code Projekt im Juni 2007 von David Cournapeau als Autor und 2010 von den französischen Forschern Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort und Vincent Michel vom Französischen Institut der Informatik in Roquencourt, Frankreich, zum öffentlichen Release gebracht. Ist es heutzutage eine der beliebtesten Python Bibliotheken für maschinelles Lernen. Es bietet eine große Auswahl von unterschiedlichsten Algorithmen und vereinfachenden Funktionen, die in dem Bereich des maschinellen Lernens häufig verwendet werden (**?**).

Gensim (Version 3.8.3): Ist eine quelloffene Programmierbibliothek, geschrieben in Python und Cython, für das Verarbeiten von textuellen Informationen und Topic Modellierung. Implementiert sind unterschiedlichste Methoden des maschinellen Lernens und statistischer Modelle. Gensim ermöglicht zudem die Verarbeitung großer Datenmengen, indem diese von der Festplatte gestreamt werden, was es von anderen Bibliotheken abhebt, die zumeist nur die Daten aus dem Arbeitsspeicher beziehen. Autor dieser Bibliothek ist Radim Rehurek und entwickelt von RARE Technologies Ltd. Die erste Version erschien 2009 (Gensim 2020).

4.2 Datenvorverarbeitung

Wichtiger Aspekt bei der Implementierung von Recommender Systemen und zeitgleich von KNNs ist die Bereinigung und Kontrolle des vorliegenden Datensatzes. In diesem Arbeitsschritt prüft man die Qualität und Struktur der Daten, anhand ausgewählter Kriterien, die zu einem beträchtlichen Teil die Qualität eben dieser Systeme widerspiegeln. Deswegen ist es sehr wichtig die bestmöglichen und dabei verfügbaren Parameter auszuwählen, die ein Szenario oder ein Objekt am besten beschreiben.
Verwendet wird der Amazon Review Datensatz (Amazon Dataset 2018). Dieser aus

Metadaten und Bewertungen bestehende Datensatz wurde von den Forschern an der University of California San Diego zusammengetragen, um eine Basis für die Forschung an Recommender Systemen zu unterstützen. Es besteht aus Daten ab Mai 1996 bis Oktober 2018 in seiner aktuellen Version und umfasst in seiner Gesamtheit 233.1 Millionen Bewertungen und Produktdaten. Dieser Datensatz unterteilt sich in verschiedene Kategorien, wie zum Beispiel Möbel, Elektronik, Musik und viele weitere. Verwendet wird die Kategorie Videogames mit ca. 87 Tausend Einträgen von Produkten. Diese Produktdaten besitzen unterschiedliche Informationen, jedoch keine nummerischen Werte, die sich für eine Weiterverarbeitung und für das Trainieren eines KNN eignen. Hierdurch werden nur die textuell vorliegenden Informationen herangezogen und auf dieser Basis ein LDA Modell trainiert. Mit der Ausgabe, die *Topic-Verteilung* pro Produkt, werden daraufhin die unterschiedlichen VAEs trainiert. Es werden alle Produkte entfernt, die keine Einträge unter den Punkten Beschreibung und Features enthalten. Nach diesem Schritt bleiben 77589 Produkteinträge übrig. Grund hierfür ist die Annahme das ein Titel allein nicht ausreicht, um Produkten eindeutige Eigenschaften zuzuordnen, was allerdings wichtig für die Weiterverarbeitung mit dem LDA Modell ist. Zudem ist die Anzahl der Wörter der bezogenen Punkte ohnehin nicht allzu groß. Im Schnitt sind pro Produkt ca. 20 Wörter verfügbar.

4.3 Latent Dirihlet Allocation Implementierung und Evaluation

Mit den gefilterten Daten aus dem vorhergehenden Schritt wird nun weitergearbeitet. Es folgen weitere Schritte bevor sie dem LDA Model als Datenbasis zur Verfügung stehen können. Die Abfolge der Schritte sind linear zu begreifen und sollten somit in dieser Abfolge eingesetzt werden. Als erstes werden die Texte *tokenisiert* und im Zuge dessen auch Sonderzeichen sowie doppelte Leerzeichen entfernt. Danach werden die Stoppwörter aus den Texten zu den Produkten entfernt. Ein Stoppwort ist ein Wort ohne Informationsgehalt wie *und, oder, somit, als* und einige weitere. Diese Stoppwortliste wurde nach einer Recherche von dem Programm-Toolkit *NLTK* bezogen wegen seiner Ausführlichkeit. Zusätzlich zu diesen Stoppwörter wurde die Liste um Wörter erweitert, welche in folgender Abbildung zu sehen sind. Nach einigen Iterationen hat sich herausgestellt das einige Produkte des Datensatzes HTML Code als Beschreibung oder Feature enthalten. So wurde versucht die meisten dieser „Code-

Worte“ manuell zu filtern. Eine Filterung wie *„lösche alle Wörter die weniger als drei Buchstaben haben“*, war ebenso nicht möglich da es in dieser Kategorie wichtige Begriffe gibt, wie zum Beispiel *„ds“*. Denn *„Nintendo DS“* ist ein Begriff mit großem Informationsgehalt im Kontext Videospiele.

```
additional_stopwords = ["ve", "games", "video", "amazon", "images", "png", "position",
                        "height", "width", "https", "padding", "com", "span", "table",
                        "font", "ps", "function", "border", "window", "border", "dsv",
                        "background", "icon", "margin", "length", "color", "text",
                        "void", "container", "left", "right", "void", "display", "na",
                        "none", "return", "image", "top", "bottom", "align", "nav", "sprite",
                        "sprites", "float", "repeat", "error", "document", "list", "id"
                        "column", "row", "uv", "else", "center", "px", "ue", "url", "ssl",
                         "ap", "javascripts", "cols", "email", "viz", "log", "td", "var",
                         "js", "tag", "lib", "popover", "td", "box", "col", "ws", "cursor",
                         "li", "ul", "th", "buy", "sc", "hasownproperty", "em", "cm", "undefined",
                         "mw", "fx", "dd", "yo", "also", "get", "however", "please", "would",
                         "want", "could", "know", "always", "may", "every", "even", "ever", "like",
                         "take", "find", "tiles", "gb", "tb", "fi", "ed" "ram", "mb", "tr", "oz",
                         "mp", "sa", "nz", "os", "lot", "hz", "weight", "time", "back", "need",
                         "made", "make", "may", "http", "www", "well", "many", "around", "help",
                         "ap", "jsexception", "header", "footer", "uelogerror", "typeof", "loglevel",
                         "ue", "csm", "size", "body", "middle", "fpf", "onerror", "settimeout", "indexof",
                         "localstorage", "redirection", "size", "eventtarget", "stub", "ctb", "isbft",
                         "attribution", "config", "ueinit", "view", "larger", "uet", "push",
                         "createelement", "onerror", "use"]
```

Abbildung 4.1: Liste zusätzlicher Stoppwörter

Nun wird ein Bi-Gramm Modell trainiert und ein Schwellwert von zwei Erscheinungen festgelegt. In diesem Kontext bedeutet das Wörter die mehr als zweimal im gesamten Textkorpus direkt nebeneinander erscheinen einen neuen Begriff bilden. Somit wird aus *„Nintendo“* und *„DS“*, *„Nintendo-DS“*. Dadurch wird das Vokabular, die Anzahl der einzelnen Worte, um die Anzahl der Bi-Gramme erweitert. Als letzter Schritt werden, falls vorhanden, leere Token gelöscht.

Üblicherweise werden in diesem Kontext Worte *lemmatisiert*. Das heißt, jedes Wort wird auf sein Grundwort reduziert. Dies soll Doppelungen vermeiden. Jedoch stellte sich heraus, dass diese Methodik spezielle Begriffe seltsam verwertet. Beispielhaft wird *„Warcraft“* zu *„wocraf“* transformiert. Somit wurde entschieden diesen Schritt nicht in die Vorverarbeitung mit aufzunehmen, denn das Problem der Doppelung ist nicht so groß mit der niedrigen Anzahl der Worte pro Produkt.

Nach diesen Schritten liegt ein für LDA verwertbarer *Bag-of-Words* Vektor für jeden Produkteintrag vor. Mit diesen Vektoren wird nun die von Gensim implementierte Version des LDA Models für zehn Epochen trainiert (Hoffman et al. 2010). Die

Standard Hyperparameter des Modells wurden nicht verändert. Das Resultat wird mit dem Tool *pyLDAvis* zugänglich gemacht. Diese Visualisierungen sind der Arbeit für diverse Mengen an Topics beigelegt.

Um die optimale Menge von Topics zu ermitteln, die das LDA Modell mit Worten füllen soll, existieren zwei Metriken die herangezogen werden können. Diese werden *Perplexity* und *Coherence* genannt. Jedoch soll laut Studien die Metrik Perplexity, welche zur Art der *predictive-likelihood* Metriken gehört, für den Menschen nicht interpretierbare Ergebnisse erzielen und haben somit beide keine Korrelation. Daher wird die Coherence Metrik verwendet (Röder et al. 2015).

Coherence: bewertet ein einzelnes Thema, indem es den Grad der semantischen Ähnlichkeit zwischen hochbewerteten Wörtern der gegebenen Topics misst. Diese Messungen helfen dabei, zwischen Themen zu unterscheiden, die semantisch interpretierbare Themen sind und Themen, die Artefakte statistischer Schlussfolgerungen sind. Eine Reihe von Aussagen oder Fakten gilt als kohärent, wenn sie sich gegenseitig unterstützen. So kann ein kohärenter Fakt in einem Kontext interpretiert werden, der alle oder die meisten Fakten abdeckt. Ein Beispiel für schlüssige Fakten ist Ës handelt sich um ein Multiplayer Spiel", Ës handelt sich um ein Ego-Shooter Spiel", În diesem Spiel treten zwei Fraktionen gegeneinander an". Es existieren einige Möglichkeiten wie Coherence gemessen werden kann, diese Arbeit verwendet das Maß **C-v** (Newman et al. 2010), (Shashank 2019).

Folgende Abbildung zeigt die Messungen zur *Coherence* dieses Datensatz. Die Skala reicht von null bis eins. Allerdings handelt es sich bei einem Wert von 0.40 bereits um ein schlechtes Resultat und Werte über 0.80 sind so gut wie nie zu erreichen. Somit wäre ein Wert von 0.70 optimal.

Wie zu sehen liegt der maximal erreichte Wert bei 0.62, was einer Topic Menge von Zehn entspricht. Bei genauerer Betrachtung zeigte sich das einige textuelle Produktdaten nur aus *HTML* oder *Javascript* Code bestehen. Dies ist in der beigefügten *pyLDAvis* Visualisierung mit zehn *Topics* zu beobachten. Folglich wurde der nächste bessere Wert gewählt, was Topic Menge 40 entspricht. Mit dieser Menge verlieren die Code Schnipsel an Gewicht in Relation zum gesamten Vokabular. Positiv ist das die Topics, welche diese Schnipsel enthalten, wenig Überlappungen mit ansons-

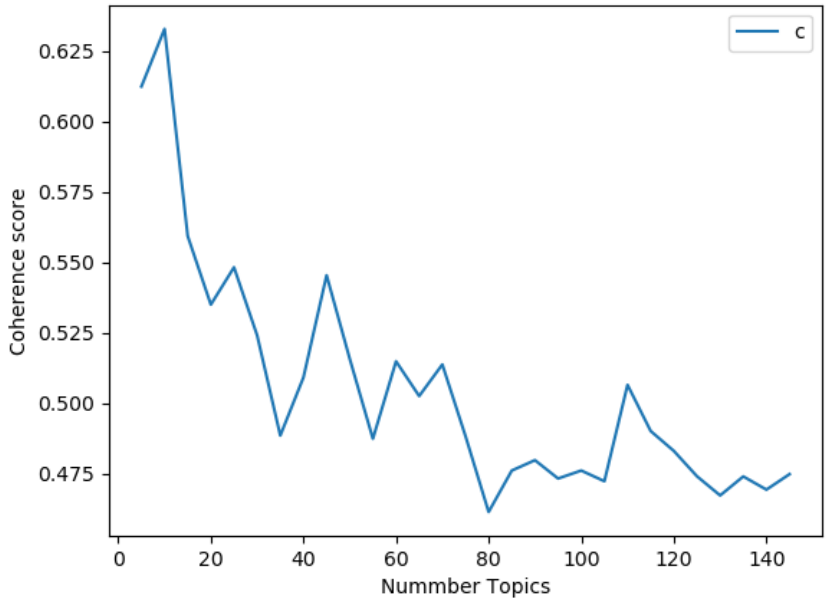

Abbildung 4.2: Coherence Graph für optimale Topic Menge

ten aussagekräftigen Topics haben. Somit können diese ein bis zwei Topics bei einer Topic-Menge über 40 ad-acta gelegt werden. Diese beiden Topics werden nicht länger berücksichtigt, der Vollständigkeit halber jedoch nicht gelöscht. Annahme hierfür ist das sonst die Aussagekraft der ursprünglichen Dimensionen Schaden nehmen würde. Als finale Topic-Menge wird 50 gewählt.

4.4 Datenbasis für den Variational Autoencoder

In diesem Schritt wird der Datensatz für den VAE vorbereitet. Hierbei wird zunächst die Topic Verteilung für jeden der 77589 Eintrage des Datensatz durch das LDA Model inferiert. Mit Hilfe dieser Verteilungen pro Produkt wird nun eine Matrix zusammengestellt. Eine Reihe entspricht einem Produkt und die Spalten repräsentieren die jeweiligen Topics. Anzumerken ist das die Positionen für die Topics nicht mit der Topic Nummer übereinstimmen. Somit sitzen die Wahrscheinlichkeiten für Topic Eins auf Position Null innerhalb der Matrix. Die resultierende Matrix besitzt die Dimension (77589×50). Bevor dieser Datensatz dem Trainingsverfahren zu Verfügung stehen kann, muss jener in Trainings- und Testdatensatz aufgeteilt werden. Man hat

sich für ein Verhältnis von 70 : 30 Prozent entschieden. Beide resultierenden Werte werden auf eine Ganzzahl abgerundet.

4.5 Implementierungen des Variational Autoencoder

In diesem Unterkapitel wird die verwendete Architektur des VAE sowie die drei verwendeten Verlustfunktionen erörtert. Die Hyperparameter werden erst im folgenden Kapitel gewählt. Verwendet wird ein VAE mit zwei verdeckten Schichten, jeweils für Encoder und Decoder. Die Anzahl der Neuronen gestaltet sich dynamisch. Die erste verdeckte Schicht besitzt 60% der Eingabe Neuronen, darauffolgend besitzt die zweite verdeckte Schicht 30% der Eingabe Neuronen. Diese Werte werden auf eine Ganzzahl abgerundet und sind symmetrisch für Encoder und Decoder. Zusätzlich werden die drei Aktivierungsfunktionen implementiert, welche im Unterkapitel *Aktivierungsfunktion* erörtert wurden. Mit *ReLU* werden die Gewichte des VAE mittels *He Normal* (He & Zhang 2015), mit *Tanh* mittels *Glorot Normal* und mit *SELU* mittels *LeCun Normal* initialisiert. Die Dimension des latenten Raums wird auf vier gesetzt. Der VAE wird in allen Szenarien 200 Epochen trainiert. Die vorgestellte Architektur wird mit drei unterschiedlichen Verlustfunktionen trainiert. Diese werden folgend erörtert.

Zusätzlich wird die Auswirkung einer Dropout-Schicht überprüft, welche noch vor der Eingabeschicht des VAE angewandt wird. Genauer gesagt handelt es bei dieser Schicht um eine spezielle Art des Dropouts, *Alpha-Dropout* genannt. Diese Variation des Dropouts, welche Durchschnitt und Varianz der Eingabe nahe den ursprünglichen Werten hält, um die selbstnormalisierende Eigenschaft auch nach dem nullen von Werten der Eingabe zu gewährleisten. *Alpha-Dropout* harmoniert mit der *SELU* Aktivierungsfunktion, da Aktivierungen nach dem Zufallsprinzip anhand des negativen Sättigungswert festgelegt werden.

4.5.1 Variational Autoencoder für die Rekonstruktion

Für die im Konzept vorgestellten Szenarien ist es nötig Vektoren, im Kontext des Datensatz, transformieren zu können. Mit Hilfe zweier unterschiedlicher Verlustfunktionen soll dies bewerkstelligt werden. Anzumerken ist das jedoch nur eine Verlustfunktion verwendet werden wird, um diese Transformationen durchzuführen. Anhand der Rekonstruktionsgenauigkeit gemessen, mittels der Kosinus-Ähnlichkeit, werden

diese beiden Funktionen evaluiert. Folgend werden beide vorgestellt.

Sigmoid-Kreuzentropie mit Logits und Kullback-Leibler Divergenz: Diese Methode misst den Wahrscheinlichkeitsfehler zwischen den wahren Einträgen des Vektors und den Logits des VAE nachdem auf diese die Sigmoid Funktion angewandt wurde (Tensorflow 2020: tf.sigmoid-cross-entropy-with-logits). Somit wird jedes Topic, bzw. jeder Eintrag des Vektors, als mögliche zugehörige Klasse wahrgenommen. Der Umstand, dass jedes Topic unabhängig und nicht gegenseitig ausschließend ist, ermöglicht dies. Zusätzlich wird die *Kullback-Leibler Divergenz* verwendet und diese mit drei unterschiedlichen β Werten multipliziert. Dies sind die Werte $0.1, 1, 2$

Sigmoid-Kreuzentropie mit Logits und Monte Carlo Simulation:

$$Verlust_{sig-mc} = log\, p(x \mid z) + log(z) - log\, q(z \mid x)\, mit\, z \sim q(z \mid x) \tag{4.1}$$

Diese Methodik verzichtet auf die *Kullback-Leibler Divergenz* und somit auch auf die Möglichkeit der Kontrolle über die Regularisierung (Tensorflow 2020). Diese verfügt jedoch über den gleichen *Rekonstruktionsterm*.

4.5.2 Multinomial-VAE für das Kollaborative Filtern

Hierbei wird zunächst die *Softmax* Funktion auf die Logits des VAE angewandt und dann mit dem Eingabevektor elementweise multipliziert. Dieser *Rekonstruktionsterm* wird mit der *Kullback-Leibler Divergenz* verwendet mit β als Koeffizient. $f(x)$ repräsentiert hierbei die *Softmax* Funktion.

$$Verlust_{mult}(x) = -\sum_i f(x_i) \cdot x_i + \beta \cdot D_{KL} \tag{4.2}$$

5 Evaluierung

In diesem Kapitel werden die Hyperparameter Anpassungen des Adam Algorithmus, die Verlustfunktionen, die drei Aktivierungsfunktionen SELU, ReLU, Tanh und die im Konzept vorgestellten Szenarien anhand ihrer Aussagekraft, Einsetzbarkeit und Performanz evaluiert. Zunächst wird der Trainingsprozess der VAEs betrachtet. Daraufhin werden die performantesten VAEs mit den unterschiedlichen Verlustfunktionen in den im Kapitel Konzept vorgestellten Szenarien eingesetzt. Da die Masse an Trainingsdurchläufen und die dazu kommenden Verlustfunktions-Graphen beträchtlich war, werden nur die bemerkenswertesten vorgestellt. Sonst würde dies zu viele Seiten beanspruchen. Jedoch können diese Graphen zu verschiedensten Durchläufen im beigefügten Dateienordner unter *model figures* betrachtet werden. Zudem erinnere man sich das ein VAE den ELBO (siehe Unterkapitel VAE Verlustfunktion, S.) versucht zu maximieren. Deshalb werden die Graphen zu den Verlusten des Trainings aufsteigende Werte zeigen und keine abfallenden wie sonst üblich, um dies zu verdeutlichen. Der Begriff *Disentanglement* in den Abbildbeschreibungen bezieht sich auf den β-Wert eingeführt durch den β-*VAE*, erläutert im Kapitel *Variational Autoencoders*. Es gilt bereits vorab zu erwähnen, dass wie erwartet, VAEs mit einem Wert von 0.1 für β bessere Verlustwerte erzielen werden. Da diese hierdurch ein höheres Augenmerk auf Rekonstruktionsgenauigkeit anstatt Regularisierung, der Kullback-Leibler Term der Verlustfunktion, legen werden.

5.1 Evaluierung des Trainingsprozesses

Betrachtet werden die Anpassungen an den Hyperparametern, die drei Aktivierungsfunktionen SELU, ReLU, Tanh und ihre Auswirkung auf den Trainingsverlauf sowie die Anwendung des Alpha-Dropouts direkt auf die Eingabedaten.

5.1.1 Variational Autoencoder für die Rekonstruktion

Die im Unterkapitel *Implementierungen des Variational Autoencoder* vorgestellten Verlustfunktionen zur Rekonstruktion von Vektoren werden hier zuerst gemäß des Trainingsverlauf betrachtet und darauffolgend anhand der Rekonstruktionsgenauigkeit verglichen. Anzumerken ist das die nummerischen Werte des Verlusts, welche Entlang der y-Achse der Graphik zusehen sind, der beiden Verlustfunktionen nicht vergleichbar sind, denn wie beschrieben, handelt es sich bei beiden um unterschiedliche Berechnungen. Es wird lediglich betrachtet werden ob diese Beiden Funktion hinzu einem globalen Optimum konvergieren. Jedoch können für sich genommen die Werte des Verlusts beider Verlustfunktionen verglichen werden. Allgemein gilt es bereits vorab Zuerst wird die *Sigmoid-Kreuzentropie mit Logits und Kullback-Leibler Divergenz* begutachtet.

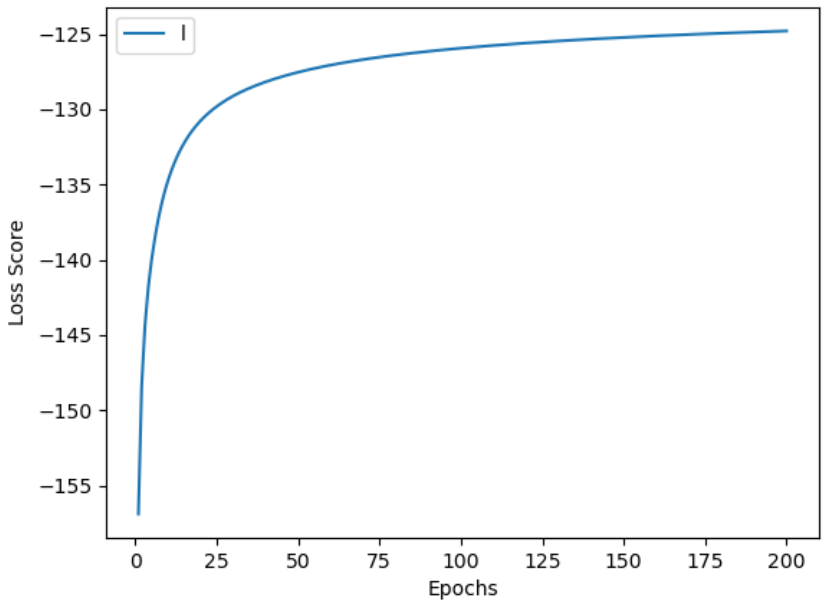

Abbildung 5.1: Sigmoid KL Divergenz, ReLU, Lernrate 0.001, Disentanglement 0.1, kein Alpha-Dropout

Die Lernrate 0.001 erwiess sich über alle Durchläufe erhaben, egal welche Aktivierungsfunktion gewählt wurde. Beispielhaft wurde 0.1, 0.01, 0.0001 sowie weitere. Wie zu sehen ist konvergieren die Aktivierungsfunktionen ReLU und SELU gegen einen ähnlich guten Werten. Wobei ReLU etwas besser abschneidet, was allerdings auf den

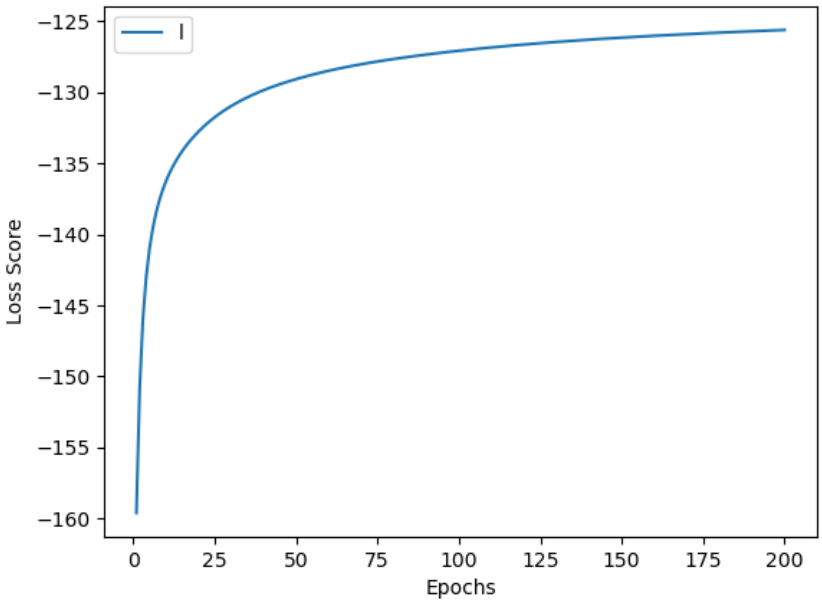

Abbildung 5.2: Sigmoid KL Divergenz, SELU, Lernrate 0.001, Disentanglement 0.1, kein Alpha-Dropout

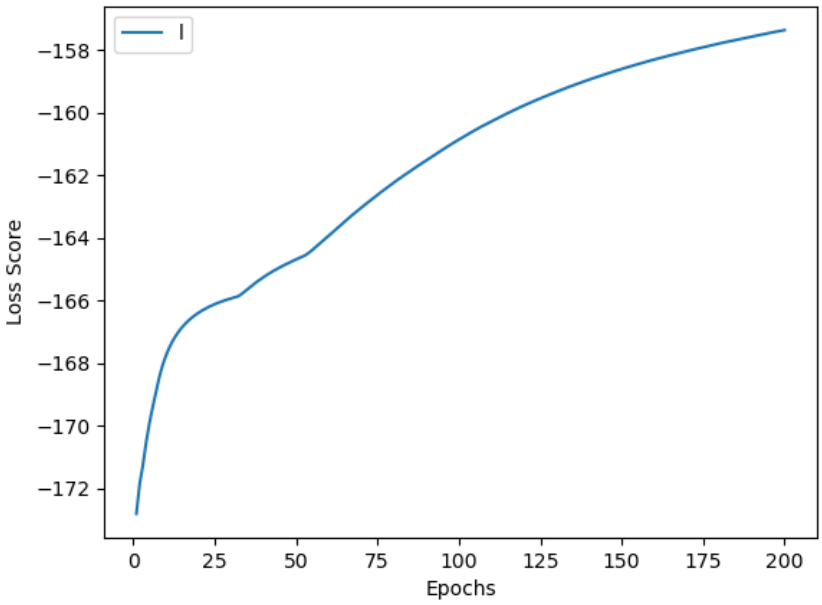

Abbildung 5.3: Sigmoid KL Divergenz, Tanh, Lernrate 0.001, Disentanglement 2, kein Alpha-Dropout

Graphen nicht ganz ersichtlich ist. Das Plus bewegt sich hierbei im Fließkommabereich bis niedrige Ganzahlen wie eins oder zwei. Tanh schneidet in allen möglichen Szenarien um einiges schlechter ab. Generell konvergieren die drei Aktivierungsfunktionen ohne die Anwendung von Alpha-Dropout mit einem Wert von 0.2 besser, was erstaunlich ist. Erwartungsgemäß hätte der VAE mit dem Einsatz von Dropout bessere Performanz bezüglich des Verlusts aufweisen. Die Annahme liegt daher nahe, dass aufgrund der vermutlichen Überzahl von Nullen in den einzelnen Vektoren der Datenbasis zu viel Informationen durch Alpha-Dropout gestrichen wird. Wie in Abbildung **5.3** zu sehen durch den unsanften Verlauf der Kurve, hat der VAE Schwierigkeiten zu konvergieren und tut dies schließlich hinzu einem lokalen Optimum.

Nun wird folgend die *Sigmoid-Kreuzentropie mit Logits und Monte-Carlo Simulation* Verlustfunktion betrachtet.

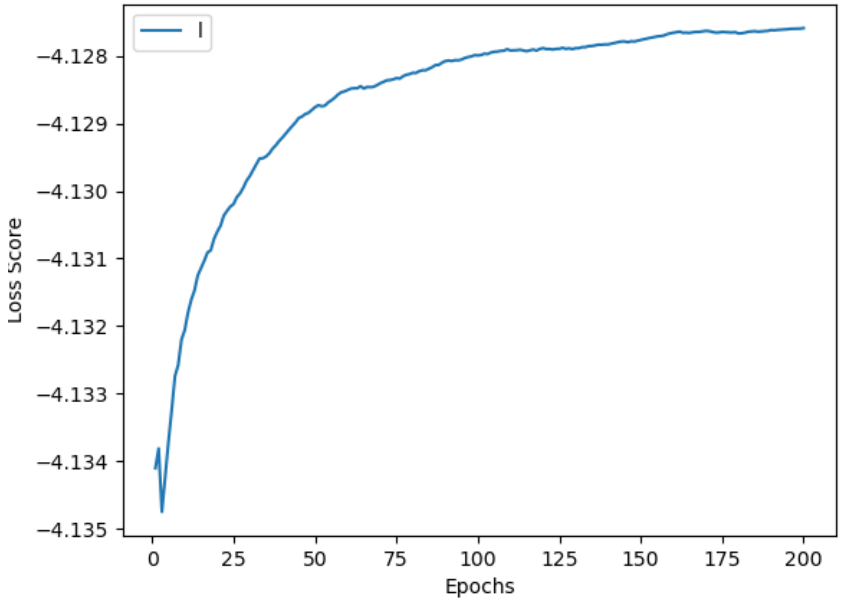

Abbildung 5.4: Sigmoid Monte-Carlo, ReLU, Lernrate 0.01, kein Alpha-Dropout

Allgemein führt die Verwendung von Alpha-Dropout bei dieser Verlustfunktion ebenfalls zu schlechteren Werten des Verlusts. Desgleichen schlägt ReLU die anderen beiden Funktionen wie zuvor. Allerdings erzielt die Lernrate 0.01 bei dieser Verlustfunktion bessere Werte.

Nun werden die beiden Funktionen mittels Rekonstruktionsgenauigkeit via Kosinus-

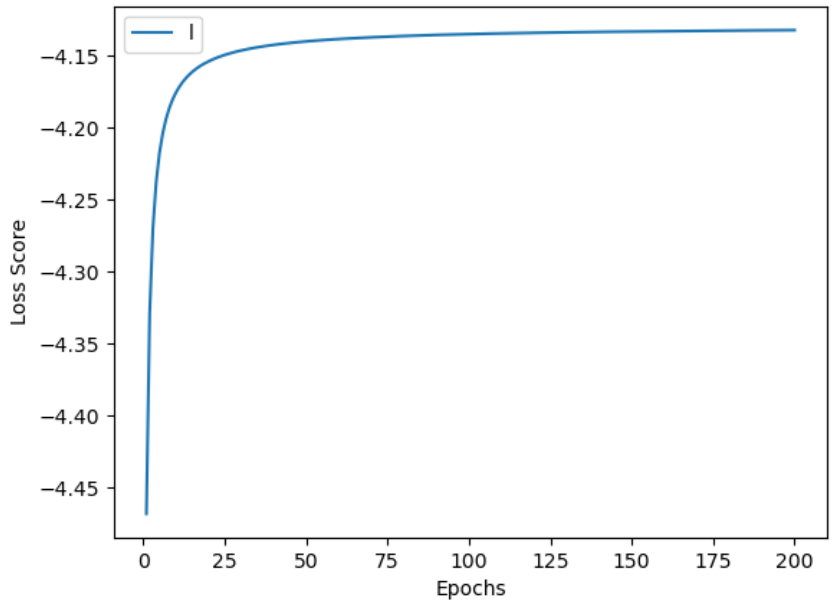

Abbildung 5.5: Sigmoid Monte-Carlo, ReLU, Lernrate 0.001, kein Alpha-Dropout

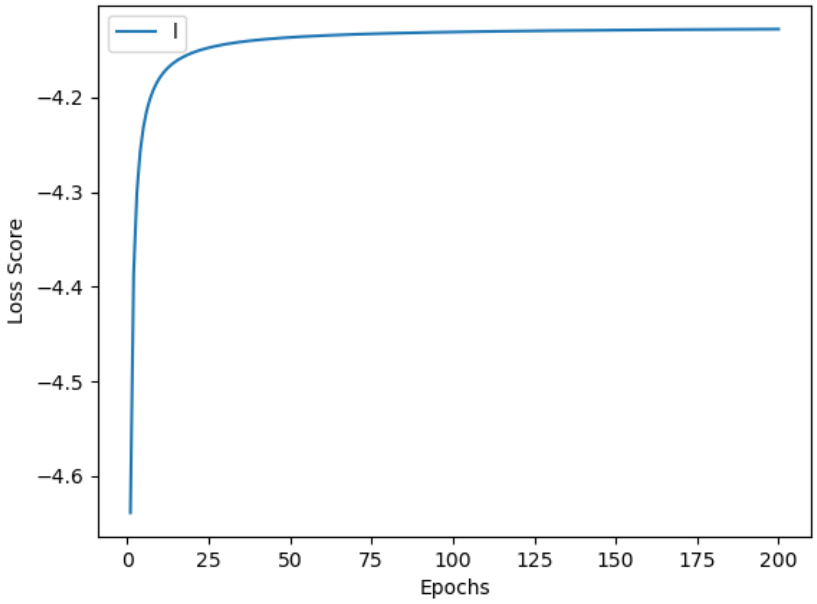

Abbildung 5.6: Sigmoid Monte-Carlo, Tanh, Lernrate 0.001, kein Alpha-Dropout

Ähnlichkeit gegenübergestellt. Bei diesem Vergleich wird der gleiche Vektor einmal durch beide VAEs mit ihren unterschiedlichen Verlustfunktionen komprimiert und rekonstruiert. Die Spalte *kos* repräsentiert die Kosinus-Ähnlichkeit. Die Werte entsprechen jeweils dem Durchschnitt von 20 zufällig ausgewählten Datenpunkten des Trainingsdatensatzes.

Platz	Verlustfunktion	kos
1	Sig-KL β=0.1	0.93
2	Sig-KL β=1	0.85
3	Sig-KL β=2	0.78
4	Sig-MC	0.44

Tabelle 5.1: Rekonstruktionsgenauigkeit via Kosinus-Ähnlichkeit

Wie bereits erwähnt, war es erwartbar das ein kleinerer β-Wert bessere Ergebnisse bezüglich der Rekonstruktionsgenauigkeit erzielt. Wie man sehen kann rekonstruiert die Variante *Sigmoid Kreuzentropie mit Kullback-Leibler Divergenz* den Eingabevektor um einiges genauer als die *Monte-Carlo* Variante. Der Abstand zwischen beiden Verlustfunktionen ist jedoch beträchtlich. Somit wird für die Aufgabe der Rekonstruktion der VAE mit Sigmoid Kullback-Leibler Divergenz, ReLU, Lernrate 0.001 und $\beta = 0.1$ gewählt.

5.1.2 Variational Autoencoder für das Kollaborative Filtern

In diesem Unterkapitel wird auch hier die Verlustfunktion anhand des Verlusts während des Trainings evaluiert. Allerdings ist der Verlustwert nicht aussagekräftig für die Anwendbarkeit im Kontext der Szenarien. Daher wird nur kurz betrachtet werden ob dieser VAE konvergiert.

Wie bereits im vorhergehenden Unterkapitel hat auch hier die Anwendung von Alpha-Dropout eine deutliche Auswirkung auf den Verlustwert. Ebenfalls schlägt ReLU die beiden anderen Verlustfunktionen. Allzeit welche Hyperparameter gewählt werden. Tanh liegt hierbei wieder hinten und SELU befindet sich wie zuvor im Mittelfeld. Desgleichen ist der Abstand zwischen SELU und Tanh wieder etwas größer als zwischen ReLU und SELU. Die optimale Lernrate liegt abermals bei 0.001.

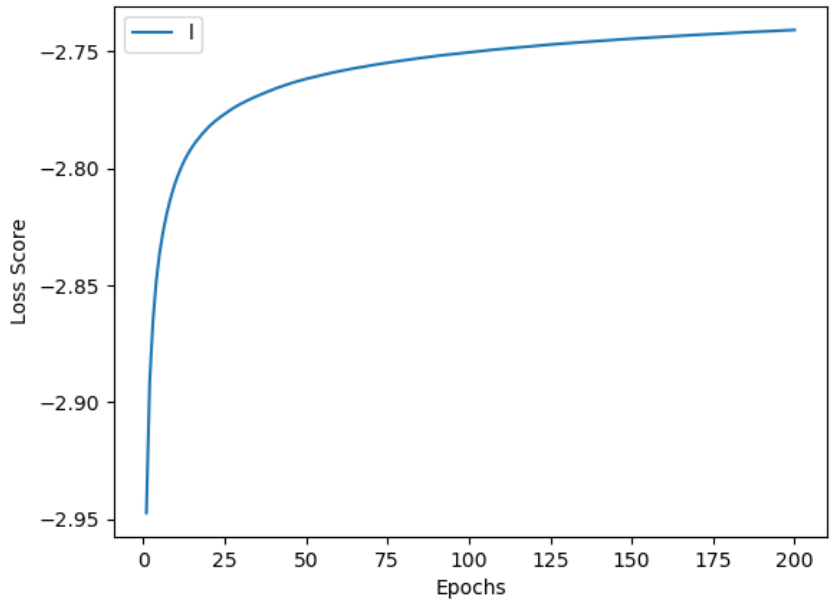

Abbildung 5.7: Multinomial-VAE, ReLU, Lernrate 0.001, Disentanglement 0.1,kein Alpha-Dropout

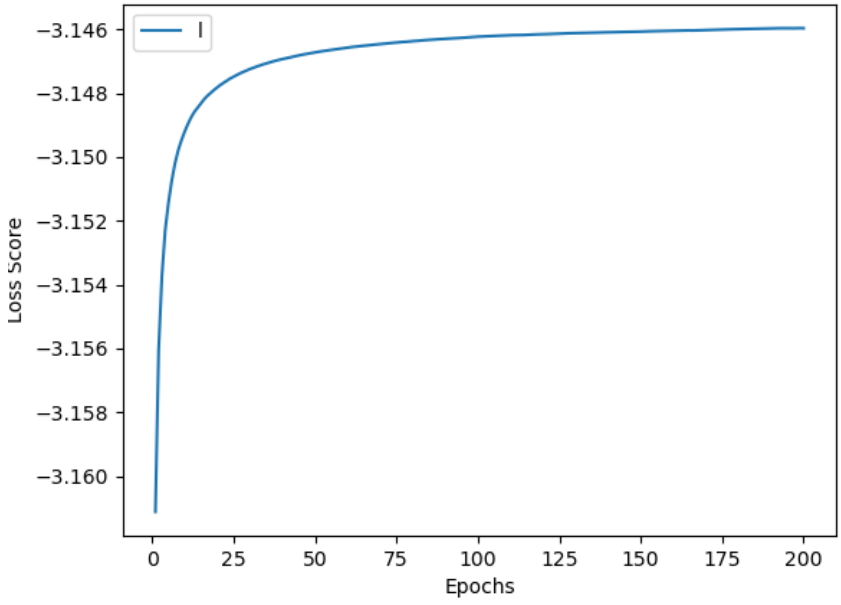

Abbildung 5.8: Multinomial-VAE, ReLU, Lernrate 0.001, Disentanglement 2,kein Alpha-Dropout

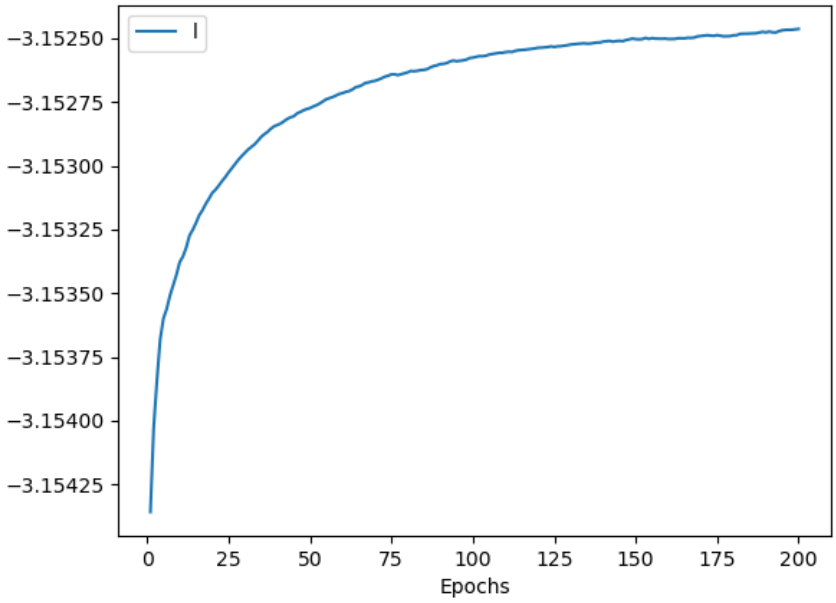

Abbildung 5.9: Multinomial-VAE, Tanh, Lernrate 0.001, Disentanglement 1,kein Alpha-Dropout

5.2 Evaluierung der Szenarien

In diesem Unterkapitel werden nun die im Kapitel *Konzept* vorgestellten Szenarien begutachtet. Die Szenarien werden anhand der Ausgabe von den einzelnen VAEs evaluiert. Hierbei werden der Übersicht halber lediglich die zehn besten Produkte geprüft, wobei die Kosinus-Ähnlichkeit die nummerische Metrik dient. Zur Erinnerung, jeder Vektor besitzt 50 Koordinaten und jene entsprechen den Topic Nummern. Die Skalare der einzelnen Koordinaten, die hierbei nicht erwähnt werden, sind als Null zu begreifen. Weiter werden alle Eingabevektoren normiert so dass sich diese im Euklidischem Raum befinden, um eine Vergleichbarkeit mit den ursprünglichen Topic Vektoren zu ermöglichen. Der Schwellwert für die Kosinus-Ähnlichkeit wird auf 0.90 gesetzt. Es wurde der Multinomial-VAE mit $\beta = 0.1$ gewählt, denn nach einigen Versuchen stellte sich herraus, dass speziell $\beta = 2$ seltsame Ergebnisse liefert. Egal welcher Vektor durch den VAE geschickt wurde, die Augabe war stehts dieselbe.

5.2.1 Szenario - 1: Kombination von Nutzer und Gegenstandvektor

Folgend wird ein Nutzervektor definiert, welcher als Basis für dieses Szenario und Szenario zwei dienen soll. Die Topics werden auf ihre entsprechenden Vektorkoordinaten gesetzt. Topic sechs mit Wertung drei und interpretiert als Action-Adventure mit Fokus auf Comic Helden. Topic acht, Wertung vier, Interpretiert als First-Person Shooter. Topic 14, Wertung 3 und wird als Sony Accessoires wahrgenommen. Topic 19, Wertung 5, soll Xbox im generellen entsprechen. Topic 23, Wertung 4, wird als Kategorie Kriegsspiele interpretiert. Topic 35, Wertung 5, soll Martial-Arts Kampfspiele betrachtet werden. Der Gegenstandstitel zu dem korrespondierenden Gegenstandsvektor lautet: Sid Meier's Civilization III PC.

Platz	Produkt	kos
1	UFC: Ultimate Fightng Championship Tapout 2 Xbox	0.99
2	Crimson Skies Xbox	0.99
3	Xbox360 Chatpad	0.99
4	Xbox360 Game Console Opening Tool	0.99
5	Forza 2 Motorsport Xbox	0.99
6	Gears of War 2 Xbox360	0.99
7	Blackwater Xbox360	0.99
8	The Bureau: XCOM Declassified	0.99

Tabelle 5.2: Produktempfehlungen nach Rekonstruktion, Nutzer und Gegestand kombiniert

In Tabelle **5.2** ist die eindeutige Dominanz des Nutzervektors über den Gegenstandsvektor zusehen. Was erfreulich ist, denn gedacht war die Kombination als Erweiterung des Nutzervektors. Dies Repräsentiert vor allem das Spiel *The Bureau: XCOM*, welches ein taktisches Third-Person Spielist. Bei *Sid Meier's Civilization* handelt es sich nämlich um ein rundenbasiertes Strategiespiel.

Die Anwendung eines *Multinomial-VAE* auf den rekonstruierten Vektor erweitert die ausgegebenen Produkte um einige Spiele aus den Bereichen Heldenepos und Action-Adventures. In gewisser Weise ist auch das Vorkommen von Xbox Accessoires positiv zu bewerten, obwohl explizit das Topic für Sony Accessoires gesetzt wurde.

Platz	Produkt	kos
1	The First Templar Xbox360	0.99
2	Hard Drive Transfer Cable Xbox360	0.99
3	Assassin's Creed Xbox360	0.99
4	Enclave Xbox Retro Gaming	0.99
5	Spiderman Xbox Retro Gaming	0.99
6	Totaled! Xbox Retro Gaming	0.99
7	Plantronics Xbox360 Headset	0.99
8	3do Control Pad	0.99
9	Fable 2 Xbox360	0.99

Tabelle 5.3: Produktempfehlungen nach Multinomial-VAE Anwendung auf die Kombination

Ebenfalls ist die Diversität der Empfehlungen als plus zu beachten.

5.2.2 Szenario - 2: Verfall von Nutzer Bewertungen

Der Nutzervektor aus dem vorgehenden Szenario soll nun einen Verfall einiger Koordinaten bzw. Bewertungen des Eingabevektors erfahren. Durch die Nutzung der im *Konzept* erörterten Funktion und zufällig ausgewählten Tagen der letzten Interaktion wird der Nutzervektor wie folgt transformiert. Topics sechs und acht bleiben unverändert bzw. drei sowie vier. Die Topics 14, 19, 23 und 35 erhalten die neue Bewertung 2.1, 4.5, 3.8 und 4.5.

Platz	Produkt	kos
1	ExcitedBots: Trick Racing with Wii Wheel Bundle	0.96
2	Trials Fusion Xbox One	0.96
3	EA Racing Pack Retro Playstation	0.95
4	GTR FIA GT Racing	0.94
5	GT Legends PC	0.94
6	NASCAR Thunder 2004 Xbox Retro	0.94

Tabelle 5.4: Produktempfehlungen durch Rekonstruktion nach Verfall der Bewertungen

Dieser Verfall einiger Bewertungen scheinen die Produktempfehlungen nur rudimentär zu ändern. Denn die Produktempfehlungen für den originellen Eingabevektor fallen ebenfalls in den Bereich Autorennspiele. So gesehen wäre dieses Verfahren brauchbar.

5.2.3 Szenario - 3: Nutzervektor mit vielen Nullen

Für dieses Szenario wird der Nutzer mit folgendem Vektor definiert. Koordinaten 11, 16, 35 werden mit den Werten 4, 3, 5 gefüllt und daraufhin normiert. Interpretiert wird Topic 11 als stark PC und Windows lastig, Topic 16 sehr stark als Autorennspiele ohne Bezug zu Konsole oder PC und Topic 35 als Richtung Martial-Arts lehnende Spiele ebenfalls ohne Bezug zu Konsole oder PC. Zusammengefasst hat dieser Vektor also drei Koordinaten und 47 Nullen. Betrachtet werden nun die Top-Ten der ähnlichsten Produkte nach der Rekonstruktion des VAE und darauf die Anwendung des Multinomial-VAE auf den rekonstruierten Vektor.

Platz	Produkt	kos
1	Slender Man Origins Mac Games	0.94
2	Moebius Demo Version PC Windows	0.94
3	Sherlock Holmes: Adventure Game Collection PC	0.94
4	Hidden Mysteries: Buckingham Palace Vampire Secrets PC	0.93
5	Black Mirror: Reiging Evil and Final Fear PC	0.93
6	Lost Chronicles: Fall of Caesar PC	0.93
7	Nancy Drew: The Deadly Device PC	0.93
8	Nancy Drew: Treasure in the Royal Tower PC	0.93
9	The Book of Desires PC	0.92
10	Mechwarrior 2 Ghost Bear's Legacy PC	0.92

Tabelle 5.5: Produktempfehlungen nach Rekonstruktion mit wenigen Einträgen

Interessanter weise zeigen beide VAEs keine Produkte an wie die Topics interpretiert wurden. Es lässt sich jedoch ein Schema in beiden finden. Das Topic *PC/Windows* ist überstark vertreten. Die Produktempfehlungen nach einer einfachen Rekonstruktion des Eingabevektors entstammen alle den Kategorien *Mystery, Horror, Thriller* und *Puzzle* Spiele. Jedoch kann keine verlässliche Aussage getroffen werden warum

Platz	Produkt	kos
1	Cradle of Egypt Collection PC Windows	0.99
2	Duck Tiles PC Windows	0.99
3	Men of War PC Windows	0.99
4	Instant Landscaping Simulation Software PC	0.98
5	Star Defender 4 PC Windowst	0.98
6	John Deere: American Farmer PC	0.98
7	Trainz Simulator Mac	0.98
8	Railroad Simulator PC	0.98
9	Harpoon Classic PC	0.97
10	Kingmaker PC Games	0.97

Tabelle 5.6: Produktempfehlungen nach Multinomial-VAE mit wenigen Einträgen

ausgerechnet diese Kategorien. Bei den Empfehlungen wie sie in Tabelle **5.6** handelt es sich beinahe ausnahmslos um Spiele aus der Kategorie *Simulation*. Hierbei sind allerdings immerhin zwei Action Spiele, *Men of War* und *Star Defender 4*, vertreten, was etwas Nachvollziehbar ist. Um einen Vergleich zu finden, wurde mittels des normierten Eingabevektors ohne VAEs nach ähnlichen Produkten gesucht. Dies führte zu keinem Resultat.

6 Fazit

Im Allgemeinen waren die Produktempfehlung durch das Anwenden der verschieden *VAEs* in einem ordentlichen Rahmen. Die Hoffnung das diese *VAEs* die latenten Beziehungen unter den einzelnen Produkten via Topic-Vektoren genau erfassen, bestätigte sich allerdings nicht. Grund hierfür kann aber auch die signifikante Diskrepanz an verfügbaren textuellen Informationen zu den einzelnen Produkten sein. Manche hatten eine Vielzahl an Worten für ihre Beschreibung zur Verfügung. Anderen hingegen lediglich Titel, Kategorie und ein paar wenige Worte. Eventuell wären diese Verfahren für Produkte besser geeignet die mehr Text enthalten. Wie zum Beispiel Zeitungsartikel, Tweets oder Blogs. Oder es sollte sichergestellt werden das alle Produkte eine ähnliche Verteilung an Worten besitzen. Für eine Vielzahl von Ad-hoc Anfragen wie sie in Online-Shops der Fall sind, eignen sich die angeführten Verfahren vermutlich nicht. Denn die Zeit zur Berechnung von Produktempfehlungen beträgt jeweils approximativ sechs Sekunden. Grund hierfür ist auch die lineare Verarbeitung der Anfrage. Somit könnten die Berechnungen der Anfragen nicht asynchron erfolgen. Jedoch könnten die Verfahren für einmal tägliche Produktempfehlungen beispielweise auf der Hauptseite eines Online-Shops dargestellt werden. Als Plus ist die Geschwindigkeit und überschaubare Trainingszeit des *LDA* Modells und den *VAE* Modellen zu nennen. In ein paar Stunden ließe sich so ein Datensatz von zigtausend Produkten bzw. ihrer textuellen Information analysieren und damit ein paar *VAEs* trainieren, mit den für diese Thesis erarbeiten Programmen.

Abbildungsverzeichnis

2.1 Kosinus-Ähnlichkeit: keine Ähnlichkeit (links), Neutralität (mittig), Ähnlichkeit (rechts) . . . 12
2.2 *Bag-of-Words* Vektoren . . . 14
2.3 Graphische Darstellung *LDA* (Blei et al. 2003) . . . 15
2.4 Beispielhafte Abbildung eines künstlichen neuronalen Netz . . . 19
2.5 Exemplarische Darstellung des Gradientenverfahren . . . 23
2.6 Adam Algorithmus . . . 25
2.7 Schematische Darstellung eines Variational Autoencoder . . . 27
2.8 Darstellung der Komprimierung duch Autoencoder und Variational Autoencoder . . . 28
2.9 Fehlerrückführung VAE . . . 31

4.1 Liste zusätzlicher Stoppwörter . . . 39
4.2 Coherence Graph für optimale Topic Menge . . . 41

5.1 Sigmoid KL Divergenz, ReLU, Lernrate 0.001, Disentanglement 0.1, kein Alpha-Dropout . . . 45
5.2 Sigmoid KL Divergenz, SELU, Lernrate 0.001, Disentanglement 0.1, kein Alpha-Dropout . . . 46
5.3 Sigmoid KL Divergenz, Tanh, Lernrate 0.001, Disentanglement 2, kein Alpha-Dropout . . . 46
5.4 Sigmoid Monte-Carlo, ReLU, Lernrate 0.01, kein Alpha-Dropout . . . 47
5.5 Sigmoid Monte-Carlo, ReLU, Lernrate 0.001, kein Alpha-Dropout . . . 48
5.6 Sigmoid Monte-Carlo, Tanh, Lernrate 0.001, kein Alpha-Dropout . . . 48
5.7 Multinomial-VAE, ReLU, Lernrate 0.001, Disentanglement 0.1,kein Alpha-Dropout . . . 50
5.8 Multinomial-VAE, ReLU, Lernrate 0.001, Disentanglement 2,kein Alpha-Dropout . . . 50

5.9 Multinomial-VAE, Tanh, Lernrate 0.001, Disentanglement 1,kein Alpha-Dropout 51

Tabellenverzeichnis

5.1 Rekonstruktionsgenauigkeit via Kosinus-Ähnlichkeit 49
5.2 Produktempfehlungen nach Rekonstruktion, Nutzer und Gegestand kombiniert 52
5.3 Produktempfehlungen nach Multinomial-VAE Anwendung auf die Kombination 53
5.4 Produktempfehlungen durch Rekonstruktion nach Verfall der Bewertungen 53
5.5 Produktempfehlungen nach Rekonstruktion mit wenigen Einträgen . . 54
5.6 Produktempfehlungen nach Multinomial-VAE mit wenigen Einträgen 55

Literaturverzeichnis

Amazon: *Amazon Produkt Dataset*, https://nijianmo.github.io/amazon/index.html, letzter Zugriff: 6. 15. 2020

Blei, David M. : *Variational Inference*, https://www.cs.princeton.edu/courses/archive/fall11/cos597C/lectures/variational-inference-i.pdf, letzter Zugriff: 6. 1. 2020

Blei, David M. & Ng, Andrew Y. & Jordan, Michael I. : *Latent Dirichlet Allocation*, http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf, letzter Zugriff: 6. 5. 2020

Duchi, John: *Derivations for Linear Algebra and Optimization*, http://stanford.edu/~jduchi/projects/general_notes.pdf, letzter Zugriff: 7. 1. 2020

Falk, Kim : *Practical Recommender Systems*, 1. Aufl., Manning Publications Co. 2019

Gensim: *Open-Source Python library for Natural Language Processing*, https://radimrehurek.com/gensim/, letzter Zugriff: 7. 22. 2020

Goldberg, David & Nichols, David & Oki, Brian & Terry, Douglas: *Using collaborative filtering to weave an information tapestry*, https://scinapse.io/papers/1966553486, letzter Zugriff: 5. 10. 2020

Goodfellow, Ian & Bengio, Yoshua & Courville, Aaron: *Deep Learning*, 1. Aufl., MIT Press 2016

He, Xiangnan & Liao, Lizi & Zhang, Hanwang : *Neural Collaborative Filtering*, https://www.comp.nus.edu.sg/~xiangnan/papers/ncf.pdf, letzter Zugriff: 5. 27. 2020

Higgins, Irina & Matthey, Loic & Pal, Arka & Burgess, Christopher & Glorot, Xavier & Botvinick, Matthew & Mohamed, Shakir & Lerchner, Alexander: *beta-VAE:*

Learning Basic Visual Concepts with a Constrained Variational Framework, https://openreview.net/forum?id=Sy2fzU9gl, letzter Zugriff: 6. 2. 2020

Hinton, Geoffrey & Salakhutdinov, Russ: „Reducing the Dimensionality of Data with Neural Networks“, *Journ. Science*, https://science.sciencemag.org/content/313/5786/504, vol. 313 (5786), 2006

Hoffman, Matthew D. & Blei, David M. & Bach, Francis: *Online Learning for Latent Dirichlet Allocation*, http://papers.nips.cc/paper/3902-online-learning-for-latent-dirichlet-allocation.pdf, letzter Zugriff: 6. 18. 2020

Hughes-Hallet, Deborah & McCallum, William G. & Gleason, Andrew: *Calculus: Single and Multivariable*, 6. Aufl., John Wiley and Sons 2012

He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian : *Delving Deep into Rectifiers:Surpassing Human-Level Performance on ImageNet Classification*, https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/He_Delving_Deep_into_ICCV_2015_paper.pdf, letzter Zugriff: 6. 20. 2020

Irhum Shafkat: *Intuitively understand Variational Autoencoders*, https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf, letzter Zugriff: 5. 15. 2020

Kingma, Diederik P. & Welling, Max: *Auto-Encoding Variational Bayes*, https://arxiv.org/abs/1412.6980, letzter Zugriff: 5. 1. 2020

Kingma, Diederik P. & Ba, Jimmy: *Adam: A Method for Stochastic Optimization*, https://arxiv.org/abs/1412.6980, Published as a conference paper at the 3rd International Conference for Learning Representations San Diego 2015, letzter Zugriff: 5. 3. 2020

Kingma, Diederik P. & Welling, Max: *An Introduction to Variational Autoencoders*, https://arxiv.org/abs/1906.02691, letzter Zugriff: 5. 26. 2020

Klambauer, Günter & Unterthiner, Thomas & Mayr, Andreas & Hochreiter, Sepp: *Self-Normalizing Neural Networks*, https://arxiv.org/abs/1706.02515, letzter Zugriff: 6. 20. 2020

Liang, Dawen & Krishnan, Rahul G. & Hoffman, Matthew D. & Jebara, Tony: *Variational Autoencoders for Collaborative Filtering*, https://arxiv.org/abs/1802.05814, letzter Zugriff: 5. 1. 2020

Murphy, Kevin P. : *Machine Learning: A Probabilistic Perspective*, 1. Aufl., MIT Press 2013

Netflix: *The Netflix Prize*, https://www.netflixprize.com/, letzter Zugriff: 5. 1. 2020

Newman, David & Lau, Jey H. & Grieser, Karl & Baldwin, Timothy: *Automatic Evaluation of Topic Coherence*, https://www.aclweb.org/anthology/N10-1012.pdf, letzter Zugriff: 6. 10. 2020

Ni, Jianmo & Li, Jiacheng & McAuley, Julian : *Justifying recommendations using distantly-labeled reviews and fined-grained aspects*, https://cseweb.ucsd.edu/~jmcauley/pdfs/emnlp19a.pdf, letzter Zugriff: 6. 15. 2020

Python: *Official Python Programming Language*, https://www.python.org, letzter Zugriff: 7. 28. 2020

Ruder, Sebastian: *An overview of gradient descent optimization algorithms*, https://arxiv.org/abs/1609.04747, letzter Zugriff: 5. 2. 2020

Röder, Michael & Both, Andreas & Hinneburg, Alexander : *Exploring the Space of Topic Coherence Measures*, http://svn.aksw.org/papers/2015/WSDM_Topic_Evaluation/public.pdf, letzter Zugriff: 6. 15. 2020

Scikit-learn: *Machine Learning library for Python*, https://scikit-learn.org/stable/, letzter Zugriff: 7. 1. 2020

Shashank Kapadia: *Evaluate topic models / Latent Dirichlet Allocation*, https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0, 2019, letzter Zugriff: 6. 15. 2020

Tensorflow: *Open-Source Neural Network library*, https://www.tensorflow.org, letzter Zugriff: 8. 1. 2020

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht.

Ort, Datum Daniel Ollhoff