Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Darius Szablowski

Vergleich verschiedener Variationaler Autencoder-Konzepte zur Generierung effektiver Molekülrepräsentationen von niedermolekularen organischen Verbindungen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Darius Szablowski

Vergleich verschiedener Variationaler Autencoder-Konzepte zur Generierung effektiver Molekülrepräsentationen von niedermolekularen organischen Verbindungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Neitzke
Zweitgutachter: Prof. Dr. Meisel

Eingereicht am: 05. Oktober 2021

Darius Szablowski

Thema der Arbeit

Vergleich verschiedener Variationaler Autencoder-Konzepte zur Generierung effektiver Molekülrepräsentationen von niedermolekularen organischen Verbindungen

Stichworte

Variationale Autoencoder, niedermolekulare organische Verbindungen, VAE, SMILES, ChemVAE, fragmentbasiertes Deep-Generatives-Model, Constrained Graph Variational Autoencoder, CGVAE

Kurzzusammenfassung

Diese Bachelorarbeit befasst sich mit verschiedenen variationalen Autoencoder-Konzepten und dem Einsatz dieser als generative Modelle für niedermolekulare organische Verbindungen. Dabei wird ausführlich auf die Grundlagen eingegangen. Es werden dabei sehr verschiedene Konzepte betrachtet, welche sich in den ihnen zugrunde liegenden neuronalen Netzwerken und sich teilweise in ihrer Vorgehensweise unterscheiden. Während eines ein Language Model für Moleküle im SMILES-Format darstellt(Gómez-Bombarelli u. a. (2018)), nutzt ein anderes unteranderem Gated Graph neural networks(Liu u. a. (2018)) und ein weiteres betrachtet Moleküle in Fragmente unterteilt als ein etwas anderes Language Model(Podda u. a. (2020)). Die Eigenschaften des Latent Spaces, der verschiedenen variationalen Autoencoder und die Eignung der verschiedenen Architekturen als generative Modelle für Moleküle werden genauer betrachtet.

Darius Szablowski

Title of the paper

Comparison of different Variational Autoencoder-Concepts for the generation of effective molecule-representations of organic small molecules

Keywords

variational autoencoders, small molecules, VAE, SMILES, ChemVAE, fragment-based deep-generative-model, Constrained Graph Variational Autoencoder, CGVAE

Abstract

This bachelor thesis looks at different variational Autoencoder concepts and their utilization as generative models for organic small molecules. This is done with a look into the underlying foundations of these topics. Specifically, very different concepts are looked at, which differ in

their underlying neural network architecture and their general approach. One is a language Model, looking at molecules in the SMILES-format charaterwise (Gómez-Bombarelli u. a. (2018)). Another one is using amongst other structure gated graph neural networks(Liu u. a. (2018)). And yet another VAE-architecture looks at molecules divided into sequences of fragments and provides a generative model for them as such(Podda u. a. (2020)). Some features of the latent spaces and the general applicability of these architecture as generative models for molecules will be investigated.

Inhaltsverzeichnis

Listings

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

Es gibt unglaublich viele kleine organische Moleküle/niedermolekulare organische Verbindungen. Laut Schätzungen sind es bis zu 10^{60} verschiedene Moleküle. Gómez-Bombarelli u. a. (2018) Um Moleküle zu finden, die einen Großteil aller Probleme, die die Menschheit heutzutage noch hat, zu lösen, bedarf es intelligenter Methoden, um diesen schier unendlich scheinenden Möglichkeitsraum an Molekülen zu erobern.

Besonders in der Entwicklung neuer Wirkstoffe(engl. *drug design*) sind generative Modelle für niedermolekulare Verbindungen gefragt. Eine Reihe neuronale Netzwerkarchitekturen wurden als generative Modelle für Moleküle vorgeschlagen. Mouchlis u. a. (2021a) Darunter gibt es mittlerweile auch eine Auswahl an variationalen Autoencoder Architekturen, welche in verschiedenen Papers als generative Modelle für Moleküle vorgestellt wurden. Podda u. a. (2020)

Um bei dieser Explosion dieser Kreativität zu wissen, und auf Basis rationaler Kriterien herauszufinden welche variationale Autoencoder Architektur sich am Besten eignet, organische Moleküle abzubilden, und als Grundlage für weitere Optimierungen von organischen Molekülen zu dienen, muss man die verschiedenen VAE-Konzepte vergleichen. Diese Arbeit soll dabei eine grundlegender Vergleich drei sehr verschiedener variationaler Autoencoder-Konzepte sein. Des Weiteren soll diese Arbeit auf Basis der betrachteten Eigenschaften des Latent Spaces, und des generativen Models des VAEs, eine erste Einschätzung dessen vornehmen, inwiefern sich die drei Konzepte unterscheiden. Darüber hinaus wird eine Bewertung vorgenommen, welches dieser Konzepte sich besonders gut als Model zur Generierung effektiver Molekülrepräsentationen eignet.

Dabei fiel die Wahl auf drei verschiedene Konzepte/Architekturen, welche in den folgenden Papers beschrieben werden: Podda u. a. (2020); Gómez-Bombarelli u. a. (2018) und Liu u. a. (2018). Die präsentierten Architekturen stellen dabei nur die Spitze des Eisberges dar, sollen aber durch ihre Verschiedenheit bei einer Einschätzung weiterer, hier nicht genannter Architekturen, hilfreich sein. Es gibt eine Klasse von generativen Modellen, welche Molekülstrukturen als

Strings(im SMILES-Format) codiert und dann zeichenweise Language Processing Modelle für diese lernt. Die in Gómez-Bombarelli u. a. (2018) vorgestellte variationale Autoencoder-Architektur ist ein Beispiel für ein solches generatives Model. Ein anderer Ansatz ist es direkt auf dem Graphen, der ein Molekül darstellt, zu arbeiten. Dafür bieten sich Graph neuronale Netzwerke an. Ein variationaler Autoencoder, der auf diesem Prinzip basiert, ist die in Liu u. a. (2018) vorgestellte Architektur. Der dritte hier vorgestellte Ansatz ist einer, welcher Moleküle(im SMILES-Format) in Fragmente unterteilt und diese wie Worte behandelt, welche zusammen einen Satz ergeben. Diese „Sätze“ werden dann von dem in Podda u. a. (2020) vorgestellten variationalen Autoencoder gelernt und generiert.

2 Grundlagen

2.1 Moleküldarstellungen

Für nicht in dem folgenden Abschnitt nicht genauer erklärte Begriffe aus der Chemie lohnt sich der Blick in das Glossar(siehe Abschnitt 6).

2.1.1 Simplified Molecular Input Line Entry System(SMILES)

Bei SMILES handelt es sich um ein chemisches Notationssystem/eine chemische Sprache, um Moleküle in eine Darstellungsform zu bringen, die für moderne Informationssysteme schneller verarbeitbar ist. Molekulare Strukturen werden mithilfe einer einfachen Grammatik, eindeutig und exakt beschrieben. Unter den verschiedenen Ansätzen zur digitalen Notation von Molekülen sind Zeilennotationen beliebt, weil sie molekulare Strukturen durch einen linearen String beschreiben, ähnlich wie natürliche Sprache. Weininger (1988)

Atome werden durch ihre Elementsymbole entsprechend dem Periodensystem codiert. Dies trägt zur intuitiven Lesbarkeit bei. Es gibt genaue Regeln, für die Notationen von anorganischen Atomen, Ionen, verschiedene Bindungstypen, Verzweigungen, Ringe im Molekül, und aromatische Strukturen. Ringe kann man sich in diesem Kontext vorstellen wie Zyklen im Molekülgraphen, welche keine weiteren Zyklen in sich enthalten. (Macrocyclische Moleküle werden in diesem Zusammenhang nicht betrachtet.)

In Abbildung 2.3 folgen einige Beispiele, die ein generelles Gefühl für das SMILES-Format vermitteln sollen. Es folgen noch ein paar Erklärungen bezüglich des SMILES-Formates, allerdings ist es für Nicht-Chemiker hilfreich erstmal eine graphische Intuition für organische Chemie und SMILES zu entwickeln, bevor man weiter liest.

Ein Beispiel für die Darstellung des Moleküls Morphin in SMILES-Notation:

O1C2C(O)C=CC3C2(C4)c5c1c(O)ccc5CC3N(C)C4

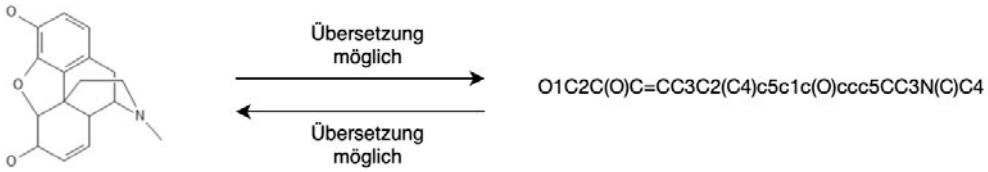


Abbildung 2.1: Eine Darstellung des Faktes, dass sich SMILES, und chemische Strukturformeln, ineinander übersetzen lassen. (Zu beachten ist bei der gezeigten Darstellung, dass das Morphin-Molekül in Wirklichkeit chiral ist, dieser Fakt wurde in dieser graphischen und dieser SMILES-Darstellung der Einfachheit halber weggelassen.) Des weiteren wurden auch die impliziten Wasserstoffatome ausgeblendet.

Source: selbstangefertigt; Die Übersetzung der SMILES, in die Strukturformel wurde mithilfe von PubChem Sketcher V2.4 vorgenommen.

Wasserstoffatome müssen in organischen Molekülen in der SMILES-Notation nicht explizit notiert werden, solange sie nach der niedrigsten normalen Valenz konsistent mit den explizit definierten Bindungen im Molekül sind.

Weininger (1988)

Wichtig zu erwähnen ist auch, dass nicht jede auf Basis des SMILES-Formats erdachte Zeichenkette unbedingt eine valide chemische Struktur darstellt. Dafür ist es nötig, diese noch zu validieren, zum Beispiel mit einem Framework, wie rdkit(http://www.rdkit.org)Landrum u. a. (2020). Kraev (2018) Ein Beispiel dafür sind Klammern, die geöffnet, aber nicht geschlossen werden. Ringe die mehrfach geöffnet, oder geschlossen werden. Im SMILES-Format werden Ringe durch ein Elementsymbol mit einer Zahl geöffnet und geschlossen. Der erste Ring im Molekül wird mit der Zahl eins geöffnet und geschlossen. Der zweite Ring im Molekül wird mit zwei geöffnet und geschlossen und so weiter.

Man kann SMILES auch als eine kontextfreie Grammatik darstellen. Kraev (2018)

Das SMILES-Format ist eine Art de facto Standard, was die die stringbasierte Darstellung von Molekülen im Machine Learning angeht. So werden alternative textbasierte Darstellungen von Molekülen in Mouchlis u. a. (2021b) nicht erwähnt. Es gibt zwar Alternativen, wie die InCHI-Darstellung(erstmals beschrieben in Heller u. a. (2013)), allerdings hat diese Darstellung sich nicht durchgesetzt im chemoinformatischen Machine Learning. Anders als bei SMILES besteht bei InCHI-Moleküldarstellungen nicht die Garantie, dass man diese wieder in Molekülgraphen umwandeln kann. Des Weiteren ist das Format nicht intuitiv für Menschen lesbar, im Gegensatz zu SMILES. David u. a. (2020)

Ein Paper(Gómez-Bombarelli u. a. (2018)), welches einen der im Folgenden betrachteten variationalen Autoencoder, beschreibt hat deutlich schlechtere Performance bei dem Versuch, einen variationalen Autoencoder mit InCHI-Molekülrepräsentationen zu trainieren. Die Autoren führen das auf die komplexere Syntax zurück, welche Zählen und Arithmetik enthält.

Initial waren SMILES nicht in der Lage, Stereochemie darzustellen. Allerdings sind sogenannte isomere SMILES mittlerweile auch in der Lage verschiedene stereochemische Isomere darzustellen. Diese Neuerung ist praktisch der neue Standard der SMILES-Nutzung. David u. a. (2020) (Vermutlich aufgrund dessen, dass stereochemisch spezifische Strukturdarstellungen unabdingbar für die organische Chemie sind.)

Für dieses Projekt stellt das SMILES-Format an erster Stelle das Datenformat da, mit dem alle betrachteten VAE-Architekturen arbeiten. Teilweise wandeln sie Moleküle im SMILES-Format in andere Datenstrukturen um, um diese zu verarbeiten.

2.1.2 Graphen-basierte Darstellung von Molekülen

Ein weiterer Ansatz zur Darstellung von Molekülen sind Graphen, und diese werden neben SMILES auch als Input für Machine-Learning-Zwecke, zum Beispiel im Drug-design eingesetzt. Mouchlis u. a. (2021b); Podda u. a. (2020)

Die Graphen-basierte Darstellung von Molekülen ist insofern offensichtlich, als das sie der intuitiven Vorstellung Molekülen gleicht, und wie man auf konzeptioneller Ebene über diese denkt. Denn in der Chemie werden die meisten organischen Moleküle, durch verschiedene Arten von Strukturformeln dargestellt. (Vollhardt und Schore, 2020, Seite 46)

Man kann sich ein Molekül als einen Graph vorstellen. Die Knoten sind die Atome, und die Kanten sind Bindungen. Natürlich gibt es bei dieser generellen Denkweise noch verschiedene

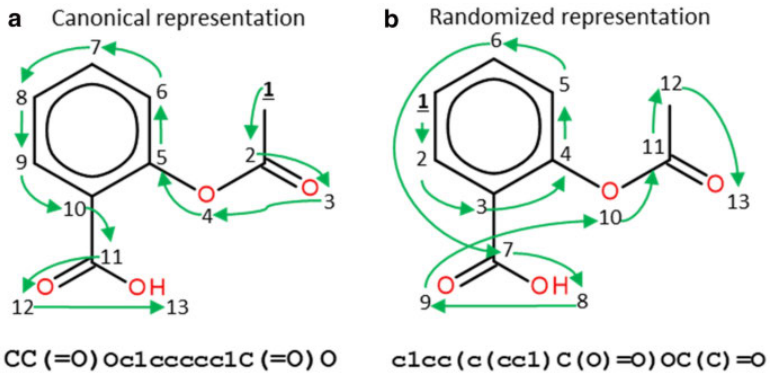


Abbildung 2.2: Diese Darstellung soll verdeutlichen, dass man abhängig von dem Atom, von dem man beginnt, ein organisches Molekül zu encoden, verschiedene SMILES-Darstellungen, für das selbe Molekül erhalten kann. Auch die Richtungen, die man wählt, um das Molekül zu encoden, verändern das SMILES-Resultat. Um dieses Problem zu umgehen, exisitieren verschiedene Kanonisierungsalgorithmen. (Das Beispielmolekül ist hier das Aspirin. David u. a. (2020))

Source: David u. a. (2020)

Arten, wie man das im Konkreten modellieren kann. David u. a. (2020)

Das in Abbildung 2.4 dargestellte Format der Graph-Kodierung stellt ein leicht vereinfachtes Beispiel dar. In realen Datensätzen sind meist größere/schwerere Moleküle enthalten, die aus mehr möglichen Atomen bestehen. Diese ließen sich nach dem generell selben Prinzip codieren. Allerdings muss man von sehr viel größeren Matrizen ausgehen, die dann zum codieren nötig sind. Des weiteren werden in dem dargestellten Format keine stereochemischen Informationen kodiert. Das bedeutet zwei verschiedene Konstitutionsisomere würden durch die selbe Codierung, in diesem Format abgebildet. Man müsste eine umfangreichere, oder mehrere Edge-Feature Matrizen nutzen, um diese Informationen mit darzustellen.

2.1.3 Fragmentbasierte Darstellungen von Molekülen

Ein Paradigma/eine Methodologie, welche sich im Bereich des *Wirkstoff-Design* entwickelt hat ist das Fragment-Based Drug Design. Podda u. a. (2020)

Fragmente sind in diesem Zusammenhang kleine strukturelle Bestandteile von Molekülen. Man kann diese Fragmente nach verschiedenen Regeln zu Molekülen kombinieren. Diese

Fragmente bestehen zumeist aus weniger, als 20 schweren Atomen. Das bedeutet Wasserstoffatome werden beim Zählen vernachlässigt. Ein Vorteil des Arbeitens mit kleineren Fragmenten ist, dass sie sich einfacher verändern lassen, als größere Fragmente. Podda u. a. (2020)

Dabei gibt es verschiedene Algorithmen, um ein gegebenes Molekül in seine Fragmente zu zerlegen. Es bietet sich in diesem Zusammenhang besonders an die Moleküle in Fragmente zu zerlegen, die besonders oft auch in anderen Molekülen vorkommen.

2.2 Moleküleigenschaften

Dieser Abschnitt befasst sich mit den Moleküleigenschaften, die wichtig für das weitere Verständnis dieser Arbeit sind. Die hier aufgeführten Eigenschaften zeichnen sich dadurch aus, dass sie berechnet werden können wenn man ihre Struktur kennt. Für den Rahmen dieser Arbeit bedeutet das konkret, dass wir diese Eigenschaften berechnen können, wenn wir den SMILES-String, der ein Molekül darstellt, kennen.

2.2.1 Synthetic Acessibility Score(SAS)

Um mithilfe von chemoinformatischen Methoden herauszufinden, wie einfach ein Stoff synthethisierbar ist, wurde der *Synthetic Accessibility Score*(deutsch. synthetischer Erreichbarkeitswert) entwickelt, dieser wurde erstmals in Ertl und Schuffenhauer (2009) beschrieben. Diese Methode basiert auf historischem Wissen aus der chemischen Synthese. Es wurden Millionen von bereits synthetisierten Molekülen analysiert, und die Komplexität der einzelnen Moleküle miteinbezogen. Diese Methode ist relativ schnell, und es besteht eine hohe Übereinstimmung zwischen händischer geschätzter synthetischer Erreichbarkeit. Diese Methode wurde entwickelt, um die Wirkstoffentwicklung(engl. drug design) zu unterstützen. Denn mithilfe dieser Methode lassen sich große Mengen an Molekülen, auf Basis ihrer Einfachheit der Synthese vergleichen. Ertl und Schuffenhauer (2009)

2.2.2 Octanol-Wasser-Partitionskoeffizient(log P)

Der Octanol-Wasser-Partitions, oder Verteilungskoeffizient ist ein Wert, der die Lipophilizität einer Verbindung beschreibt. Amézqueta u. a. (2020) Er ist definiert als das logarithmierte

Verhältnis zwischen der Löslichkeit einer Substanz in einer flüssigen Octanol-Phase, und der Substanz in einer flüssigen Wasser-Phase. Moldoveanu und David (2015)

$$logP = log\left(\frac{\text{Löslichkeit}_{Wasser}}{\text{Löslichkeit}_{Octanol}}\right) \tag{2.1}$$

(Die Löslichkeit ist als Stoffmengenkonzentration angegeben.)
Moldoveanu und David (2015)

Der Octanol-Wasser-Verteilungskoeffizient wird mit $logP$ oder K_{ow} abgekürzt. Moldoveanu und David (2015)

Dieser Wert ist sehr wichtig für die Bestimmung darüber, wie sich Moleküle in biologischen Systemen verhalten. Des weiteren ist dieser Wert in der Wirkstoffentwicklung sehr wichtig. Amézqueta u. a. (2020)

Es exisistieren Programme, die den $logP$ für gegebene Moleküle, auf Basis ihrer Struktur, approximieren können. Moldoveanu und David (2015)

2.2.3 Quantitative Estimate of Drug-Likeness(QED)

QED entspricht auf Deutsch in etwa Quantitative Abschätzung der Wirkstoff-Ähnlichkeit.

Die generelle Ähnlichkeit eines Moleküls zu bereits existierenden Wirkstoffen/als Medikamenten eingesetzten Molekülen, ist ein wichtiges Kriterium in der Wirkstoffentwicklung. Das Paper Bickerton u. a. (2012) präsentiert einen quantitativen Wert, mit dem man die Drug-Likeness eines Moleküls ermitteln kann. Zur Evaluation wurden acht physikochemische Eigenschschaften von Molekülen, zu einer speziellen Funktion zusammengefasst. Der Wertebereich reicht von null(keine der Eigenschaften sind im wünschenswerten Bereich), zu eins(alle Eigenschaften sind im wünschenswerten Bereich). Im Vergleich zu regelbasierten Ansätzen, stellt der QED-Wert ein nuancierteres Bild der *Drug-Likeness* dar, dass anstatt die Erfüllung einzelner Regeln mit Einsen und Nullen zu bewerten, diese durch einen Gesamtwert mit weichen Übergängen darstellt. Bickerton u. a. (2012)

2.3 Neuronale Netzwerke(Generell)

2.3.1 Perceptrons

Eine Art der künstlichen Neuronen im Machine Learning sind sogenannte Perceptrons. Ein Perceptron bekommt mehrere binäre Inputs und produziert daraus einen binären Output. Es

wird dabei eine Gewichtung der Inputs vorgenommen. Übersteigt die gewichtete Summe: $\Sigma_j\ w_j x_j$, einen gegebenen Schwellwert(engl. threshold value), dann gibt das Perceptrons für die gegeben Inputs, 1 aus, tut sie das nicht gibt das Neuron 0 aus(siehe Formel 2.2). (Nielsen, 2015, Seite 2 - 3)

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases} \tag{2.2}$$

(Nielsen, 2015, Seite 3)

Das Zeichen j stellt dabei die Anzahl, und den Index der Inputs und Gewichte da. Für j Inputs **x** gibt es entsprechend j Gewichte **w**.

Diese Darstellung lässt sich weiter in eine andere, aber inhaltlich äquivalente Schreibweise umformen. Als erstes können wir $\sum_j w_j x_j$, auch als punktweises Produkt, der Vektoren **x** und **w** darstellen, da gilt: $w \cdot x \equiv \sum_j w_j x_j$. Des weiteren können wir auf den $w \cdot x$ Term, noch einen sogenannten Bias-Term addieren: $w \cdot x + b$. Diesen definieren wir wie folgt: $b \equiv -\text{threshold}$. Dadurch ist die zusammengefasste Form der Umformungen(Formel 2.3) äquivalent zu Formel 2.2. Der threshold-Term wurde dabei nur auf die andere Seite der Ungleichheit verschoben. (Nielsen, 2015, Seite 4)

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \tag{2.3}$$

(Nielsen, 2015, Seite 4)

2.3.2 Perceptrons mit Aktivierungsfunktion

Perceptrons selber stellen nur eine vereinfachte Version dessen dar, was im Machine Learning als künstliches Neuron heute genutzt wird. Der Output eines Neurons wird hier normalerweise eine reelle Zahl sein. Des weiteren handelt es sich bei Gewichten und den Inputs normalerweise auch um reelle Zahlen. Der Term $w \cdot x + b$, den wir bereits im Abschnitt 2.3.1 kennegelernt haben, wird als Input für die Aktivierungsfunktion eines Neurons genutzt: (Aggarwal, 2018, Seite 5 - 6)

$$output = \sigma(w \cdot x + b) \tag{2.4}$$

(Aggarwal, 2018, Seite 5 - 6) (Nielsen, 2015, Seite 8)

Einige Aktivierungsfunktionen beispielhaft:

- Step Function: Formel 2.3 (Nielsen, 2015, Seite 4, Seite 9)
- Die Sigmoid-Funktion: $\sigma(x) = \frac{1}{1+e^{-x}}$ (Nielsen, 2015, Seite 8)
- Der Tangens hyperbolicus: $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ (Nielsen, 2015, Seite 122)
- Rectified Linear Unit(Abkürzung: ReLU): $ReLU(x) = max(x, 0)$ (Aggarwal, 2018, Seite 13)
- Softmax: $Softmax(v_i) = \frac{e^{v_i}}{\sum_j e^{v_j}}$ mit j ist die Anzahl aller Neuronen in dem betrachteten Layer; (wird über alle Neuronen eines Output Layers definiert)(Aggarwal, 2018, Seite 14)

2.3.3 Feed-Forward neuronale Netzwerke

(Andere Bezeichnungen: - Deep feedforward networks - multilayer perceptrons(MLPs) - Feed-Forward neuronale Netzwerke - Feed-Forward neuronale Netze) Feed-Forward neuronale Netzwerke(engl.: Feed-Forward Neural Networks) sind die grundlegende Netzwerk-Architektur im Bereich des *Deep Learning*.

Das Ziel eines Feed-Forward-Netzwerkes ist es eine Funktion f^* zu approximieren. Ein beispielhafter Classifier $y = f^*(\mathbf{x})$ lernt einen Input $\mathbf{x}$ auf eine Kategorie $\mathbf{y}$ zu mappen.

Ein Feed-Forward neuronales Netz definiert sich das Mapping $\mathbf{y} = f(\mathbf{x}, \boldsymbol{\theta})$ und lernt dann die Werte der Parameter von $\boldsymbol{\theta}$ die dazu führen, dass $f(\mathbf{x}, \boldsymbol{\theta})$ die Funktion $f^*(\mathbf{x})$ am besten approximiert.

Diese Art von neuronalen Netzwerken, werden als Feed-Forward Netzwerke bezeichnet, weil die Informationen, durch das Netzwerk nur nach vorne fließen(i.e. es gibt keine Feedback-Verbindungen, oder Schleifen).

(Goodfellow u. a., 2016, Seite 164 - 165)

Feed-Forward Netzwerke sind extrem wichtig für Machine Learning. Sie sind die Grundlage, auf denen viele neuronale Netzwerkarchitekturen konzeptionell aufbauen.

Sie werden als **Netzwerke** bezeichnet, weil sie typischerweise aus vielen verschiedenen Funktionen aufgebaut sind. Es handelt sich dabei um einen azyklischen gerichteten Graphen, der beschreibt wie die Funktionen genau miteinander kombiniert werden. Diese Funktionen sind ihrerseits jeweils in verschiedene Lagen(engl. Layer), aufgeteilt. Die Funktionen können dabei nur von einem Layer zum nächsten verbunden werden. Das erste Layer nach dem Input, wird als erstes Layer bezeichnet, und so weiter. Die Anzahl der Layer im Netzwerk wird als Tiefe(engl.) des neuronalen Netzwerks bezeichnet. Das ist auch, wo der Begriff *Deep Learning* herrührt. Das letzte Layer, eines Feed-Forward Netzwerkes wird als **Output Layer** bezeichnet. Während des Trainings des neuronalen Netzwerkes versuchen wir $f(\mathbf{x})$ an $f^*(\mathbf{x})$ anzunähern.

Die Trainingsdaten bestehen typischerweise aus verschiedenen ungenauen Beispielen, für die Ergebnisse von $f^*(\mathbf{x})$, an verschiedenen Punkten. Die Trainingsdaten bestehen aus Paaren, aus Punkten $\mathbf{x}$, mit ihrem dazugehörigen Label $\mathbf{y} \approx f^*(\mathbf{x})$. Die Trainingsdaten sind damit eine direkte Spezifikation darüber, was das Output-Layer ausgeben soll, wenn das Netzwerk die Input-Daten $\mathbf{x}$ bekommen hat. Es soll einen Wert ausgeben, welcher nah an $\mathbf{y}$ ist. Das Verhalten der anderen Layer ist nicht über die Trainingsdaten spezifiziert. Der Lern-Algorithmus, muss sich entscheiden, wie er diese Layer nutzt, um den erwünschten Output zu generieren. Aufgrund dessen, das die Trainingsdaten, nicht spezifizieren, was diese Layer tun sollen, werden sie als **hidden Layers** bezeichnet. (Goodfellow u. a., 2016, Seite 165)

Diese Netzwerke werden als **neuronale** Netzwerke bezeichnet, weil sie lose von Neurowissenschaften inspiriert sind.

Jedes hidden Layer ist typischerweise durch einen Vektor dargestellt. Die Dimensionalität der hidden Layer bestimmt die Breite(engl. width) des Netzwerkes. Jedes Element dieses Vektors spielt eine Rolle, die sich mit der eines Neurons vergleichen lässt.

Konzeptionell kann man Layer, sowohl als eine Vektor-zu-Vektor Funktion nachdenken, aber auch als eine Reihe unabhängiger Einheiten, die parallel arbeiten. Jedes Element des Vektors kann man dann schematisch, wie ein Neuron sehen, unter dem Gesichtspunkt, dass es Input von vielen anderen Einheiten bekommt, und daraus seine eigene Aktivierung berechnet. Die Idee, dass man mehrere Layer bestehend aus vektorwertigen Repräsentationen verwendet kommt aus den Neurowissenschaften. Die Wahl der der Funktionen $f^{(i)}(\mathbf{x})$, welche genutzt werden, um die Aktivierungen zu berechnen sind auch lose inspiriert von Beobachtungen darüber, wie biologische Neuronen Berechnungen anstellen. Die Moderne Forschung an neuronalen Netzwerken ist basiert vor allem auf verschiedenen mathematischen

und ingenieurwissenschaftlichen Disziplinen, und das Ziel neuronaler Netzwerke im Machine Learning ist es nicht, perfekt das Gehirn abzubilden. (Goodfellow u. a., 2016, Seite 165)

2.3.4 Loss Funktionen

Um ein neuronales Netzwerk zu trainieren benötigt man einen Algorithmus, welcher in der Lage ist, die Werte für die Gewichte und Biases genau so zu finden, dass das Netzwerk $f^*(x)$ gut für alle Trainingsinputs x approximiert. Um das zu quantifizieren definiert man sich eine Loss-Funktion(auch Cost-Funktion oder Objective-Funktion genannt). (Nielsen, 2015, Seite 16) Ein Beispiel für eine solche Cost-Funktion ist die sogennante mittlere quadratische Abweichung(engl. mean squared error(mse)):

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2 \tag{2.5}$$

In diesem Zusammenhang:

- *w* steht für alle Gewichte in einem neuronalen Netzwerk
- *a* steht für die Outputs eines neuronalen Netzwerkes
- *x* steht für den Input
- *y(x)* steht für die auf Basis der Trainingsdaten erwarteten Outputs

(Nielsen, 2015, Seite 16)

2.3.5 Gradient Descent

Generell

Ziel des Trainings eines neuronalen Netzwerkes mithilfe des Gradient Descent-Algorithmus ist es, möglichst globale Minima der Loss-Funktion zu finden.(Kubat, 2017, Seite 97) Dafür müssen wir Werte für die Gewichte w und die Biases b finden, so dass die Loss-Funktion möglichst klein ist. Dies tun wir indem wir wiederholt die folgenden Update-Regeln für diese verwenden:

$$w_k \rightarrow w_k' = w_k - \eta \frac{\partial C}{\partial w_k} \tag{2.6}$$

$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial b_l}. \tag{2.7}$$

(Nielsen, 2015, Seite 22)

- Der Ausdruck w_k steht in diesem Zusammenhang für die Gewichte in Layer k
- Der Ausdruck b_l steht in diesem Zusammenhang für die Biases in Layer l
- Das Zeichen η steht in diesem Zusammenhang für die **Lernrate(engl. learning rate)**. Dieser Wert stellt eine Gewichtung für die partiellen Ableitungen der Cost-Funktion dar.(Nielsen, 2015, Seite 19) Dieser Wert ist ein kleiner positiver Parameter. Er stellt schematisch dar wie groß die Schritte sind, die wir auf Basis der letzten Ableitung(en) in Richtung des globalen Minimums gehen. Sind die Werte dafür zu klein, dauert das Training länger, als nötig. Sind die gewählten Werte für die Lernrate allerdings zu groß ist es möglich, dass man das globale Minimum verfehlt.
- Auf die Berechnung der Terme $\frac{\partial C}{\partial w_k}$ und $\frac{\partial C}{\partial b_l}$ wird in 2.3.6 genauer eingegangen werden.

Stochastic Gradient Descent und Mini-Batch Verfahren

Ein Problem, welches man beim Einsatz des Gradient Descents hat, ist dass man sehr viele Werte berechnen muss. Hat man als Cost-Funktion eine Summe von $C = \frac{1}{n} \sum_x C_x$ verschiedenen Werten, mit $C_x = \frac{\|y(x)-a\|^2}{2}$(mit dem Beispiel des mean squared errors). In der Praxis muss man dafür den Gradienten ∇C_x für jeden einzeln Trainingsinput berechnen. (Die Komponenten von ∇C_x sind alle Werte von allen Layern, aus $\frac{\partial C}{\partial w_k}$ und $\frac{\partial C}{\partial b_l}$ zusammengefasst, allerdings nur für einen Trainingsdurchgang.) Dann müsste man die Werte summieren und einen Durchschnitt bilden, für alle einzelnen enthaltenen Werte($\nabla C = \frac{1}{n} \sum_x \nabla C_x$). Damit würde man dann die konkreten Werte haben, die man in die vorher beschriebenen Update-Regeln einsetzen könnte. In der Praxis ist die Menge der Werte, die man dafür zu berechenen muss sehr groß und der gesamte Trainingsprozess geht deswegen nur sehr langsam vonstatten. (Nielsen, 2015, Seite 22)

Um dieses Problem zu lösen verwendet man eine Abwandlung des Gradient Descents, namens **Stochastic Gradient Descent**. Dabei wählt man zufällig eine kleine Menge an Inputs, der Größe m, aus den Trainingsdaten. Diese bezeichnen wir dann, als ein *Mini-batch*. Dies wird unter der Annahme getan, dass der durchschnittliche Gradient, der mithilfe des kleineren Anteils der Trainingsdaten berechnet wurde, sehr nah an dem durchschnittlichen Gradienten aller Trainingsdaten liegen wird. (Nielsen, 2015, Seite 22)

Daraus ergeben sich folgende Update-Regeln:

$$w_k \to w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k} \tag{2.8}$$

$$b_l \to b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l} \tag{2.9}$$

(Nielsen, 2015, Seite 22)

=> Das sind alle partiellen Ableitungen eines Durchgangs, in Abhängigkeit aller Komponenten eines Feed-Forward-Durchganges. In 2.3.6 wird darauf detaillierter eingegangen.

Dann wird wieder ein Mini-Batch ausgewählt und der Prozess wiederholt. Das wird getan bis alle Trainingsinputs aufgebraucht sind. Ist das der Fall hat man eine *Epoche(engl. epoch)*, des Trainings beendet. (Typischerweise macht man beim Training des neuronalen Netzwerkes mehrere Traingsepochen.) (Nielsen, 2015, Seite 23)

Die in den Formeln 2.8 und Formel 2.9 dargestellten Summenterme beziehen sich auf die komponentenweisen Summen der Gradienten, jeweils über alle Gradienten, für jedes der Elemente im Trainingsdatensatz. Das Ganze wird mit $\frac{1}{m}$ skaliert, um zu verdeutlichen, dass man hier nur einen Anteil der Trainingsdaten abbildet. (Es existieren allerdings verschiedene Konventionen dazu, wie man skaliert. Dies stellt nur eine Möglichkeit dar.)(Nielsen, 2015, Seite 23) Das führt mathematisch dazu, dass ein einzelnes Minibatch, die Parameter(Gewichte und Biases) eines Netzwerkes nicht direkt überproportional verändert.

2.3.6 Backpropagation

Ein Algorithmus zur Berechnung des Gradienten der Loss-Funktion/der Steigung dieser im multidimensionalen.

Der englische Begriff lässt sich wörtlich, als Rückführung/Zurückpropagierung übersetzen. Dabei wird der Error(i.e. die Abweichung der erhaltenen, mit den erwünschten Outputs), durch die Loss-Funktion erhalten, durch das neuronale Netzwerk, auf die einzelnen Gewichte und Biases zurückgeführt. Damit ist konkret gemeint, dass man guckt, wie viel Einfluss, die einzelnen Gewichte und Biases, auf die Abweichung vom gewünschten Output, haben. Dies wird mithilfe einer partiellen Ableitung der Loss-Funktionen, einmal in Bezug auf die einzelnen

Gewichte($\partial C/\partial w$), und einmal in Bezug auf die einzelnen Biases($\partial C/\partial b$).(Nielsen, 2015, Seite 43)

$$\delta^L = \nabla_a C \odot \sigma'\left(z^L\right) \tag{2.10}$$

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right) \tag{2.11}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{2.12}$$

$$\frac{\partial C}{\partial w^l_{j\,k}} = a^{l-1}_k \delta^l_j \tag{2.13}$$

(Nielsen, 2015, Seite 47)

- Das Zeichen C beschreibt den Output der Loss Funktion. Ein anderer Name für die Loss-Funktion, ist Cost-Funktion, woher das große C kommt.
- Das Zeichen L steht für das letzte Layer, und gleichzeitig für die Anzahl aller Layer.
- Das Zeichen l steht für das Layer Nummer l.
- Das Zeichen a beschreibt eine oder mehrere Aktivierungen/Outputs
  - In der Formel 2.10, steht das a für die Aktivierungen des letzten Layers.
  - In der Formel 2.13, steht das a^{l-1}_k für eine Aktivierung(i.e. den Output), des Layers vor l, des Neurons Nummer k.
  - Die Aktivierung für ein Neuron im Layer l, Nummer j, ist wie folgt definiert:

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right) \tag{2.14}$$

- Das σ steht für die verwendetet Aktivierungsfunktion, beispielsweise Sigmoid.(Das σ' steht entsprechend für die erster Ableitung dieser Funktion.)
- Das Zeichen $\odot$ beschreibt das punktweise Produkt, auch elementweises Produkt.
- Der Ausdruck δ^l bezeichnet den Error für ein Layer Nummer l.

- Der Ausdruck δ^L bezeichnet den Error für das letzte Layer.
- $\nabla_a C$ steht für einen Vektor, dessen Komponenten alle partiellen Ableitungen $\frac{\partial C}{\partial a_j^L}$ sind. Vereinfacht gesagt beschreibt dieser Term die Änderungsrate der Cost-Funktion im Bezug auf die Output-Aktivierungen.
- Das hoch T, steht für das Transponieren einer Matrix.
  - Im konkreten Fall, in Formel 2.11 wird mit dem Term $(w^{l+1})^T$ die Gewichtsmatrix, des Layers l+1 transponiert.
- Das Zeichen *z* steht wie *a*, auch für eine Ausgabe.
  - Der Ausdruck z^L steht für die Outputs des letzten Layers.
  - Der Ausdruck z^l steht für die Outputs des l-ten Layers.
- Das Zeichen *w* steht für ein Gewicht, oder mehrere Gewichte.
  - Der Ausdruck w^{l+1} steht für alle Gewichte, des Layers l + 1.
  - Der Ausdruck $w^l_{j\,k}$ steht für das Gewicht, welches das Neuron Nummer k, aus dem (l - 1)ten-Layer mit dem Neuron Nummer j, aus dem l-ten-Layer verbindet
- Das Zeichen b steht für ein Bias.
  - Der Ausdruck b^l_j steht für ein konkretes Bias, in Layer l, des Neurons Nummer j.

Die Bedeutung der Formeln zusammengefasst:

- Die Formel 2.10, beschreibt die Formel, mit welcher sich der Error für das letzte Layer berechnen lässt.
- Die Formel 2.11, beschreibt die Formel, mit welcher sich der Error für alle Layer, außer dem letzten Layer berechnen lässt.(Mit Ausnahme des ersten Layers, dieses wird auch als Input Layer bezeichnet, und die Aktivierungen dieses Layers sind durch das konkrete Element der Trainingsdaten gesetzt.)
- Die Formel 2.12, und damit der Ausdruck $\frac{\partial C}{\partial b^l_j}$ beschreibt, wie sehr sich die Cost-Funktion durch eine Veränderung eines konkreten Bias b^l_j verändert. Das Ergebnis wird als δ^l_j bezeichnet.

- Die Formel 2.13, und damit der Ausdruck $\frac{\partial C}{\partial w^l_{j\ k}}$ beschreibt, wie sehr sich die Cost-Funktion durch eine Veränderung eines konkreten Gewichts $w^l_{j\ k}$ verändert. Das Ergebnis ist gleich des Ergebnisses, der Multiplikation der Aktivierung(i.e. dem Output des vorigen Layers) mit dem in Formel 2.12 definierten δ^l_j.

(Nielsen, 2015, Seite 43 - 47)

Interessant an diesem Algorithmus ist vor allem, dass alle nötigen Schritte einfach zu berechnen sind, denn die Ableitung der Cost-Funktion nach a lässt sich im Vorhinein einfach berechnen. Die Ableitung der Aktivierungsfunktion auch. Da bietet sich sogar an bei dem Feed-Forward Durchgang gleichzeitig, nebenbei $\sigma'\left(z^L\right)$ zu berechnen. Das macht man natürlich in der Praxis mit allen Werte gleichzeitig, in Vektoren oder Matrizenform, für jeden Schritt.

Ein kleines Beispiel, mit einer konkreten Cost-Funktion: Eine oft eingesetzte Cost-Funktion ist die mittlere quadratische Abweichung.:

$$C = \frac{1}{2}\|y - a^L\|^2 = \frac{1}{2}\sum_j (y_j - a^L_j)^2, \tag{2.15}$$

Die Ableitung nach allen Aktivierungen ist dann: $\nabla_a C = (a^L - y)$. Diese Formel lässt sich jetzt, in Formel 2.10 einsetzen, und dadurch sind die restlichen Berechnungsschritte, mit den vorangegangenen Erklärungen, ersichtlich.

1. Input x: Setze die Aktivierungen a^1 für das Input-Layer.
2. Feedforward: Berechne für jedes Layer l = 2,3,...,L: $z^l = w^l a^{l-1} + b^l$ und $a^l = \sigma\left(z^l\right)$
3. Output Error berechnen δ^L: $\delta^L = \nabla_a C \odot \sigma'\left(z^L\right)$
4. Den Error zurückverbreiten(Backpropagating): $\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right)$
5. Output: Der Gradient der Cost-Funktion wird gegeben durch: $\frac{\partial C}{\partial b^l_j} = \delta^l_j$ und $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$

(Nielsen, 2015, Seite 49)

In der Praxis wird Backpropagation meist, mit Gradient Descent(siehe Abschnitt 2.6) kombiniert. Das kombinierte Vorgehen ist beschrieben in 3:

1. Input x: Eine Menge an Trainingsdaten.
2. Für jedes Element in den Trainingsdaten:
   - Feedforward: für jedes Layer l = 2,3,...,L berechne: $^{x,l} = w^l\, a^{x,l-1} + b^l \, und \, a^{x,l} = \sigma\left(z^{x,l}\right)$
   - Output Error: Berechne den Vektor: $\delta^{x,L} = \nabla_a C_x \odot \sigma'\left(z^{x,L}\right)$
   - Backpropagate the Error: Berechne für jedes Layer l = L-1, L-2, ..., 2: $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$
3. Gradient descent: Verändere die Gewichte und Biases, für jedes Layer l = L, L-1, . . . , 2; nach den folgenden Regeln:

   Für die Gewichte: $w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$
   Für die Biases: $b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$

(Nielsen, 2015, Seite 49 - 50)

Möchte man jetzt Stochastic Gradient Descent(siehe Absatz 2.3.5) nutzen braucht man noch zusätzlich eine äußere Schleife, welche Minibatches generiert. Des weiteren bräuchte man für die praktische Anwendung des Algorithmus 3 eine äußere Schleife, die dafür zuständig ist, den Traingsprozess für alle Trainingsepochen zu wiederholen. (Nielsen, 2015, Seite 50)

Der hier dargestellte Backpropagation-Algorithmus basiert auf der Kettenregel, aus der Infinitesimalrechnung. (Nielsen, 2015, Seite 50)

2.4 Autoencoder

2.4.1 Generell

Autoencoder sind eine neuronale Netzwerkarchitektur mit dem Ziel dass die Output Daten möglichst dem Input gleichen. Die Input-Daten in den Output zu kopieren mag im ersten Moment nutzlos erscheinen. Allerdings ist man bei Autencodern typischerweise nicht direkt interessiert

am Output des Encoders. Stattdessen hofft man, dass das Training des Autencoders dazu führt, dass der Output des Encoders nützliche Eigenschaften annimmt. (Goodfellow u. a., 2016, Seite 499)

Die Architektur besteht aus zwei Teilen:

- einer Encoder Funktion $\mathbf{h} = f(\mathbf{x})$
- einer Decoder Funktion $\mathbf{r} = g(\mathbf{h})$; welcher Originaldaten aus dem Output des Encoders rekonstruieren soll

(Goodfellow u. a., 2016, Seite 499)

Wobei x der Input ist, und h ein encodeter Vektor der Inputdaten. Natürlich ist es dafür nötig, dass der Encoder Teil aus mindestens einem Layer bestehen. Autencoder kann man als Spezialfall eines Feedforward-Netzwerks betrachten, dass sich mit den selben Trainingsmethoden trainieren lässt. Diese sind typischerweise minibatch Gradient descent, wobei diese auf den Steigungen basieren, die mit dem Backpropagation-Algorithmus berechnet wurden.(Goodfellow u. a., 2016, Seite 499 bis 500)

Das Traininieren eines Autencoders lässt sich als Kompromiss folgender zwei Kräfte/Ziele betrachten:

1. Das Lernen einer Repräsentation h eines Trainingsbeispiels, so dass x ungefähr rekonstruiert wird, wenn man h als Input in den Decoder hineingibt.
2. Das Erfüllen einer Bedingung(engl. constraint) oder einer Regularisierung. Das kann eine architekturelle Beschränkung sein(zum Beispiel: eine bestimmte Anzahl an Layers, oder $|h| < |x|$), oder es kann der Einsatz eines Regularisierungsterms in der Loss Function sein. Der Einsatz dieser Methoden führt zu einem Model/Autencoder, das eher unabhängig von den Input-Daten ist.

(Goodfellow u. a., 2016, Seite 513)

Das reine Kopieren des Input in die Outputneuronen ist nicht besonders nützlich, aber ein Modell, welches unabhängig vom Input ist, auch nicht. Nur durch die Zusammenarbeit dieser Kräfte, werden diese nützlich, weil sie den Autencoder dazu zwingen, dass die versteckte Repräsentation(h) der Daten, wichtige Eigenschaften der Gesamtheit der Trainingsdaten lernt. (Goodfellow

u. a., 2016, Seite 500 bis 501)

Traditionelle Einsatzgebiete von Autencodern sind Dimensionalitätsreduktion und das Lernen von Eigenschaften(engl. feature learning), von Datensätzen. (Goodfellow u. a., 2016, Seite 502)

Wenn man Autencoder zur Dimensionalitätsreduktion nutzt, dann nutz man ein internes Bottleneck-Layer gibt. Dieses besteht aus weniger Knoten, als die Input- und Output-layer. Dieser Umstand zwingt das Netzwerk bei dem Training dazu kompakte Datenrepräsentationen zu lernen. Diese Form der Autencoder nennt man *undercomplete Autencoders*. (Goodfellow u. a., 2016, Seite 500 bis 501)

Wenn der Decoder linear ist(eine lineare Aktivierungsfunktion hat), und als Loss-Funktion die mittlere quadratische Abweichung(engl.: mean squared error(abgekürzt: mse)), dann lernt ein undercomplete Autoencoder den selben Raum abzudecken, wie die Hauptkomponentenanalyse(engl.: principal component analysis(abgekürzt: PCA)). In diesem Fall hat der Autencoder, der auf das Kopieren der Daten vom Input in den Output trainiert wurde, als Nebenwirkung den Hauptkomponentenraum der Daten gelernt. (Goodfellow u. a., 2016, Seite 500 bis 501)

Autencoder mit nicht linearen Encoder-Funktionen und nicht linearen Decoder-Funktionen können also mächtigere nicht lineare Generalisierung der Daten lernen als PCA. (Goodfellow u. a., 2016, Seite 522)

Eine neuere Entwicklung im Bereich der Autencoder ist es die Encoder und Decoder-Funktionen nicht mehr als deterministische, sonder als stochastische Mappings umzusetzen: $p_{encoder}(\mathbf{h}|\mathbf{x})$ und $p_{decoder}(\mathbf{x}|\mathbf{h})$. Der Encoding- und Decoding Prozess lassen sich dann als das nehmen eine Stichprobe aus einer Wahrscheinlichkeitsverteilung betrachten.(Goodfellow u. a., 2016, Seite 506 bis 507)

2.4.2 Variationale Autoencoder(VAEs)

Normale Autoencoder lassen sich in ihrer Grundform nicht als generative Modelle nutzen, da ihr Latent Space nicht kontinuierlich ist, und keine Interpolation erlaubt. (Der Latent Space ist der Merkmalsraum, den ein Autoencoder mit seinem Encoder-Teil generiert.) Shafkat (2018)

Variationale Autoencoder haben weniger mit einfachen Autoencodern zu tun als man im ersten Moment vermuten würde. Anstatt dass man versucht den Input zu einem festen Vektor im Latent Space zu mappen, ist die Idee bei einem variationalen Autoencoder, dass man den Input auf eine Wahrscheinlichkeitsverteilung mappen möchte. Weng (2018)

Wir bezeichnen diese Verteilung als p_θ parametrisiert durch θ. Dann kann man die Beziehungen, zwischen den Inputdaten $\mathbf{x}$ und dem kodierten Vektor im Latent Space $\mathbf{z}$, wie folgt beschreiben:

- die A-Priori-Wahrscheinlichkeitsverteilung: $p_\theta(\mathbf{z})$
- Wahrscheinlichkeitsverteilung $p_\theta(\mathbf{x}|\mathbf{z})$
- die A-Posteriori-Wahrscheinlichkeitsverteilung: $p_\theta(\mathbf{z}|\mathbf{x})$

Weng (2018)

Unter der Annahme, dass wir die tatsächlichen Parameter für θ^* kennen. Dann müsste man, um einen reales Element des Originaldatensatzes zu generieren:

1. Einfach eine Stichprobe aus der A-Priori-Verteilung $p_{\theta^*}(\mathbf{z})$ nehmen(engl. sample).
2. Mit diesen Parametern könnten wir dann mithilfe der bedingten Wahrscheinlichkeit $p_{\theta^*}(\mathbf{x}|\mathbf{z} = \mathbf{z}^{(i)})$, einen realen Eintrag $\mathbf{x}^{(i)}$ aus den Originaldaten generieren.

Weng (2018)

Die optimalen Parameter θ^* sind die, die Wahrscheinlichkeit maximieren, dass man mit ihnen Realdaten/Originaldaten generieren kann. Als Formel ausgedrückt bedeutet das:

$$\theta^* = \arg\max_\theta \prod_{i=1}^{n} p_\theta(\mathbf{x}^{(i)}) \tag{2.16}$$

In der Praxis arbeitet man meist mit den logarithmierten Wahrscheinlichkeiten:

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \log p_\theta(\mathbf{x}^{(i)}) \tag{2.17}$$

Zusammengefasst sieht dann die generative Formel, für einen Dateneintrag $\mathbf{x}^{(i)}$ aus dem Latent Space $\mathbf{x}$ wie folgt aus:

$$p_\theta(\mathbf{x}^{(i)}) = \int p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z} \tag{2.18}$$

In der Realität ist es schwer $p_\theta(\mathbf{x}^{(i)})$ auf diesem Weg zu berechnen, da man alle mögliche Werte für $\mathbf{z}$ ausprobieren müsste, und diese dann aufaddieren müsste. Um das zu umgehen braucht man eine Funktion $q_\phi(\mathbf{z}|\mathbf{x})$, welche mit gegebenem $\mathbf{x}$ in der Lage ist, uns $\mathbf{z}$ zu generieren. Diese Funktion ist parametrisiert mit ϕ. Weng (2018)

- Die bedingte Wahrscheinlichkeit $p_\theta(\mathbf{x}|\mathbf{z})$ beschreibt den probabilistischen Decoder.
- Die Näherungsfunktion $q_\phi(\mathbf{z}|\mathbf{x})$ stellt den probabilistischen Encoder dar.

Weng (2018)

Die geschätzte A-Posteriori-Wahrscheinlichkeitsverteilung $q_\phi(\mathbf{z}|\mathbf{x})$ soll so nah wie möglich an der realen A-Posteriori-Verteilung $p_\theta(\mathbf{z}|\mathbf{x})$, wie möglich sein.

Kullback-Leibler Divergenz Ein Maß dafür, wie verschieden zwei Verteilungen sind ist die Kullback-Leibler Divergenz(auch als KL-Divergenz bezeichnet). Weng (2018)

$$D_{KL}(p||q) = \int_x q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \tag{2.19}$$

Die Formel 2.19 beschreibt die Kullback-Leibler-Divergenz D_{KL} zwischen den Wahrscheinlichkeitsverteilungen p und q.

Erdogan (2017)

Evidence Lower Bound(ELBO) Die Loss-Funktion von VAEs heißt *evidence lower bound*(abgekürzt ELBO)(Kingma und Welling, 2019, Seite 16), auf Deutsch untere Beweisgrenze.

In unserem Fall wollen wir die KL-Divergenz $D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$ minimieren.Weng (2018)

Durch einige Umformungsschritte erhalten wir:

$$log\ p_\theta(x) = \mathbb{E}_{q_\phi(z|x)}[log\ p_\theta(x)] \tag{2.20}$$

$$= \mathbb{E}_{q_\phi(z|x)}\left[log\ \left[\frac{p_\theta(x,z)}{p_\theta(x|z)}\right]\right] \tag{2.21}$$

$$= \mathbb{E}_{q_\phi(z|x)}\left[log\ \left[\frac{p_\theta(x,z)}{q_\phi(z|x)}\frac{q_\phi(z|x)}{p_\theta(x|z)}\right]\right] \tag{2.22}$$

$$= \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[log\ \left[\frac{p_\theta(x,z)}{q_\phi(z|x)}\right]\right]}_{=\mathcal{L}_{\theta,\phi}(x)\ (ELBO)} + \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[log\ \left[\frac{q_\phi(z|x)}{p_\theta(x|z)}\right]\right]}_{=D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x}))} \tag{2.23}$$

(Kingma und Welling, 2019, Seite 15 bis 18)

Der Umformungsschritt von Formel 2.20 zu Formel 2.21 ist möglich, da gilt:

$$p_\theta(x) = \frac{p_\theta(x,z)}{p_\theta(z|x)} \tag{2.24}$$

Weng (2018)

Der Umformungsschritt von Formel 2.21 zu Formel 2.22 ist möglich, wenn man den inneren Bruch um $q_\phi(z|x)$ erweitert.

Der Umformungsschritt von Formel 2.22 zu Formel 2.23 ist möglich, durch Anwendung der Logarithmusgesetze.

$log\ p_\theta(x)$ wird auch als die Evidenz bezeichnet. Die Bezeichnung *Evidence Lower Bound* kommt daher, dass die KL-Divergenz immer größer gleich 0 ist, und ELBO und die KL-Divergenz in folgender Beziehung zueinander stehen $\mathcal{L}_{\theta,\phi}(x) = logp_\theta(x) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x}))$; ist die ELBO immer die untere Grenze für die Log-Wahrscheinlichkeit, der Originaldaten. (Kingma und Welling, 2019, Seite 18)

Also ist die Loss Formel wie folgt:

$$\mathcal{L}_{\theta,\phi}(x) = log\ p_\theta(\mathbf{x}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) \tag{2.25}$$

(Kingma und Welling, 2019, Seite 18)

Durch diese Loss-Formel erreicht man, dass:

1. Die Wahrscheinlichkeit $p_\theta(\mathbf{x})$ maximiert wird => also wird unser generatives Model besser.
2. Des weiteren erreicht man, dass die Approximierung $q_\phi(\mathbf{z}|\mathbf{x})$ näher an der wahren A-Posteriori-Wahrscheinlichkeitsverteilung $p_\theta(\mathbf{z}|\mathbf{x})$ ist, was bedeutet dass unser Encoder besser wird.

(Kingma und Welling, 2019, Seite 19)

Durch weitere Umformungen, auf die in diesem Rahmen nicht weiter eingegangen wird, erhält man die folgende Form der Loss-Formel, des variationalen Autoencoders:

$$\mathcal{L}_{\theta,\phi}(x) = -\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} log\ p_\theta(\mathbf{x}|\mathbf{z}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) \tag{2.26}$$

Weng (2018)

Der erste Term, auf der rechten Seite der Formel stellt die Rekonstruktionswahrscheinlichkeit eines Elements aus dem Datensatzes dar. Der zweite Term besagt, wie nah unser gelernter probabilistischer Decoder an unserer erwünschten Verteilung des Latent-Spaces $p_\theta(\mathbf{z})$ ist. Für $p_\theta(\mathbf{z})$ setzen wir fest, dass eine gaußsche Standardnormalverteilung ist. Jordan (2018)

Setzt man das Ganze jetzt für ein neuronales Netzwerk um wird der erste Term zum Reconstruction-Loss, und der zweite zur Summe der KL-Divergenzen, aller Dimensionen, zwischen den empirisch ermittelten Werten für μ und σ, und denen einer Standardnormalverteilung($\mu = 0$ und $\sigma = 1$). Jordan (2018)

$$\mathcal{L}_{\mathrm{VAE}} = \mathcal{L}(x, \hat{x}) + \sum D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) \tag{2.27}$$

Jordan (2018)

Reparametrization Trick Um mithilfe von $q_\phi(\mathbf{z}|\mathbf{x})$ den Latent Space Vektor $\mathbf{z}$ zu ermitteln, müssen wir Stichproben mithilfe der durch $q_\phi(\mathbf{z}|\mathbf{x})$ generierten multidimensionalen Model-Parameter μ und σ nehmen. Weng (2018); Shafkat (2018) Eine Stichprobe nehmen/sampeln ist ein stochastischer Prozess. Durch eine zufallsabhängige Formel, lässt sich keine Backpropagation vornehmen. Es ist in diesem Fall möglich die Zufallsvariable $\mathbf{z}$, mithilfe einer Hilfsvariable, als deterministische Variable darzustellen. Die Hilfsvariable ist hier ϵ. Damit das möglich ist müssen wir $\mathbf{z}$ transformieren. Weng (2018)

Für eine multivariate gaußsche Normalverteilung sieht diese Transformation, wie folgt aus:

- $\mathbf{z}$, als stochastische Variable

$$\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) = \mathcal{N}(z; \mu^{(i)}, \sigma^{2(i)}\mathbf{I}) \tag{2.28}$$

- $\mathbf{z}$, als deterministische Variable; Stochastizität in ϵ ausgelagert

$$\mathbf{z} = \mu + \sigma \odot \epsilon, \text{ mit } \epsilon \sim \mathcal{N}(0, \mathbf{I}) \tag{2.29}$$

Genau diese Umformulierung von $\mathbf{z}$, damit Backpropagation durch diese Formel möglich ist, wird als Reparametrization Trick bezeichnet.

Zusammenfassung-Architektur Die Abbildung 2.11 stellt die finale Architektur des variationalen Autoencoders dar, welche aus den vorherigen Erklärungen hervorgeht. Der probabilistische Encoder $q_\phi(\mathbf{z}|\mathbf{x})$ generiert μ und σ, jeweils mit der Dimensionalität des Latent Spaces. Diese beiden Vektoren werden genutzt um den Latent Vektor, mithilfe einer Standardnormalverteilung zu sampeln, für jede Dimension. Der probabilistische Decoder $p_\theta(\mathbf{x}|\mathbf{z})$ nutzt dann $\mathbf{z}$, als Input um den $\mathbf{x}$ zu rekonstruieren. Weng (2018)

2.5 Rekurrente Neuronale Netzwerke

2.5.1 Generelles

Rekurrente neuronale Netzwerke(engl.: Recurrent Neural Networks, Abk.: RNNs) sind eine Netzwerkstruktur, die sich besonders für die Verarbeitung von Sequenzdaten eignet. Dies erreichen sie insbesondere dadurch, dass die Inputdaten, schrittweise, mehrfach in die selbe neuronale Netzwerkstruktur, gegeben werden. Gleichzeitig entwickelt sich der interne Zustand, der rekurrenten neuronalen Netzwerke in jedem Zeitschritt. Dieser interne Zustand wird auch als Hidden State bezeichnet, und dient zusammen mit den Inputdaten, für jeden zeitlichen oder logischen Schritt, als ein weiterer Input. Wichtig zu beachten ist dabei, dass der hidden state des vorigen Schrittes(t-1), als Input für den aktuellen Schritt(t) genommen wird. Die Weitergabe der Informationen, die im Hidden state enthalten sind, unterscheidet sich zwischen verschiedenen Konzepten rekurrenter neuronaler Netzwerke. (Goodfellow u. a., 2016, Seite 367 bis 372); Olah (2015)

Man kann sich das Ganze vorstellen, wie ein Netzwerk mit Schleife(siehe Abbildung 2.12).

Diese Schleife kann man sich vorstellen, wie eine Kette, die das selbe neuronale Netzwerk mehrfach enthält. Dieses erhält jetzt in jedem Zeitschritt einen neuen Input, und den Hidden State, des vorigen Schrittes, wie bereits erwähnt.Olah (2015)

In dem in den Abbildung 2.12, 2.13 und 2.14 beleuchteten Beispiel, wird der Tangens hyperbolicus, als Aktivierungsfunktion genutzt. Ansonsten ist noch erwähnenswert, dass der Hidden-State und der schrittabhängige Input konkateniert werden, bevor sie als Input für den nächsten Schritt des RNNs genutzt werden.Olah (2015)

Man unterscheidet dabei zwischen verschiedenen Typen von RNNs:

(a) Rekurrente neuronale Netzwerke, welche einen Output in jedem Zeitschritt erzeugen, und Verbindungen zwischen den Hidden-Units haben.

(b) Rekurrente neuronale Netzwerke, welche einen Output in jedem Zeitschritt erzeugen, und den Output an den nächsten Schritt, als Input für den Hidden-State geben, und nicht den Hidden-State selber. Die ist insofern möglicherweise nachteilig, weil dadurch möglicherweise weniger Informationen weitergegeben werden. Dies kann zum einen durch die möglicherweise geringere Dimensionalität des Outputs(in jedem zeitlichen Schritt), der Fall sein, aber auch dadurch, dass die Werte des Hidden States, durch ihre tatsächlichen Werte mehr, für den folgenden Zeitschritt wichtige Informationen haben.

(c) Rekurrente neuronale Netzwerke, welche rekurrente Verbindungen zwischen den Hidden Units haben, und für eine Sequenz nur Output im letzten zeitlichen Schritt ausgeben.

(Goodfellow u. a., 2016, Seite 372)

Die zugrunde liegenden Formeln, für ein beispielhaftes RNN, in Verbindung mit Abbildung 2.15:

$$a^{(t)} = b + W * h^{(t-1)} + U * x^{(t-1)} \tag{2.30}$$

$$h^{(t)} = tanh\left(a^{(t-1)}\right) \tag{2.31}$$

$$o^{(t)} = c + V * h^{(t)} \tag{2.32}$$

$$\hat{y}^{(t)} = softmax\left(o^{(t)}\right) \tag{2.33}$$

(Goodfellow u. a., 2016, Seite 374)

Diese Formeln beschreiben ein einfaches beispielhaftes RNN, mit der Aktivierungsfunktion tanh. Die Variablen b und c stellen biases dar. U, V und W stellen Gewichtungen dar. Des weiteren wird ein Post-processing Schritt vorgenommen. Der Output o(t) wird zusätzlich noch in eine Softmax-Funktion gegeben. Das kommt einem tatsächlichen Einsatz von RNNs besonders nah, da diese oft in Sequence-Generation Tanks eingesetzt werden.(Goodfellow u. a., 2016, Seite 374) Wenn man sich aus einer Liste von möglichen nächsten Elementen entscheiden muss, bietet sich die Softmax-Funktion deswegen an, weil sie dafür sorgt, dass alle Output-Werte zusammen 1 ergeben, wodurch die Werte automatisch voneinander abhängig sind, und Szenarios mit vielen hohen Zahlenwerten vermieden werden können.

2.5.2 Das Problem mit Long Term Dependencies

Ein mathematische Problem, was sich beim Arbeiten mit rekurrenten neuronalen Netzwerken ergibt, ist dass diese nicht besonders gut darin sind long term Dependencies/weit voneinander entfernte Abhängigkeiten in den Eingabedaten zu erkennen. Dies ergibt sich daraus, dass Gradienten, welche über mehrere Stufen weiter übertragen werden, meistens verschwindend klein werden, oder in sehr seltenen Fällen unerwartet explodieren (das passiert selten, ist aber fatal für den Optimierungsprozess des neuronalen Netzwerkes.) Selbst wenn man davon ausgeht, dass das rekurrente Netzwerk Erinnerungen(z.B im Hidden State) stabil speichern kann und die Gradienten nicht explodieren, werden die Gewichtungsveränderungen, die weit entfernten Abhängigkeiten beigemessen werden, weitaus geringer ausfallen als die für sehr nahe Abhängigkeiten. Das liegt daran, dass ein Abweichungswert, der am Ende der Loss-Funktion berechnet wird, bei der Durchführung des Backpropagation-Algorithmus durch die wiederholte Multiplikation mit den Deltas und Gewichtungen voriger Layer multipliziert, und so immer kleiner wird. (Goodfellow u. a., 2016, Seite 396 bis 399)Olah (2015)

2.5.3 Long Short Term Memory Networks

Bei Long Short Term Memory Networks(Abk. LSTM), erstmal beschrieben in(Hochreiter und Schmidhuber (1997)), handelt es sich um eine auf RNNs aufbauende neuronale Netzwerkarchitektur. Diese wurde mit dem expliziten Ziel entwickelt, gleichermaßen gut mit weit

entfernten(engl. long term), als auch mit näheren(engl. short term) Abhängigkeiten(engl. dependencies), umgehen zu können.(Goodfellow u. a., 2016, Seite 404)Olah (2015) Die wörtliche Übersetzung des Begriffes, Langzeit-Kurzzeit-Gedächtnis-Netzwerke, deutet schon darauf hin.

Das grobe Konzept ist das gleiche wie bei RNNs, nur dass man den Hidden State jetzt auf zwei Wegen, von einem zeitlichen/logischen Schritt an den anderen gibt.

Diese graphische Darstellung soll eine Einführung, in die interne Struktur von LSTMs geben:
Auf den ersten Blick sieht die LSTM-Netzwerken zugrunde liegende Architektur, selbst mit englischer Legende erst einmal ein wenig kompliziert aus. Doch die einzelnen Elemente werden, im folgenden schematisch, und mathematisch betrachtet und erklärt werden. Zu dem in RNNs bereits beschriebenen Hidden-State kommt in der LSTM-Architektur noch ein Cell-State hinzu(siehe Abbildung 2.17).

Dieser wird im Gegensatz zum Hidden State deutlich weniger, nur durch einfache lineare Transformationen, mathematisch verändert. Diese wie Gatter wirkenden Transformationen dienen als eine Methode selektiv Informationen, von einem Schritt in den nächsten weiterzuleiten.Olah (2015)

Der in Abbildung 2.18 dargestellte Teil einer Zelle konkateniert den Hidden-State des vorigen Schrittes, zusammen mit den Input-Daten des aktuellen Schrittes. Diese Daten werden dann gewichtet, mit zugehörigen Biases, in die Sigmoid-Aktivierungsfunktion übergeben. Schematisch dient das Ergebnis dieser Funktion f_t dazu, zu beeinflussen, welcher Teil des Cell States, sich gemerkt werden soll, und welcher vergessen werden soll. Dies wird dadurch gewährleistet, dass alle möglichen Ausgabewerte der Sigmoid-Funktion im Intervall zwischen 0 und 1 sind. Diese Ausgabewerte werden, wie in Abbildung 2.18 und Abbildung 2.20, dargestellt, danach mit dem Cell State multipliziert. Das sorgt dafür, dass Werte nahe 0 bewirken, dass Informationen, an einer Stelle in der Matrix oder dem Vektor, welche den Cell State darstellen, aus dem Cell State vergessen werden sollen. Ein Wert nahe 1, sorgt entsprechend dafür, dass die Informationen erhalten bleiben sollen, und in den nächsten Schritt übertragen werden sollen. Der in Abbildung 2.18 beschriebene Teil des LSTMs, wird zusammen mit der darauf folgenden Multiplikation mit dem Cell State, als Forget Gate bezeichnet.Olah (2015)

Darauf folgt der Teil der LSTM-Zelle, welcher darüber bestimmt, welche neuen Informationen, dem Cell State hinzugefügt werden sollen, und welche nicht.

Der in $\tilde{C}_t$ berechnete Wert stellt die neu gelernten Informationen dar. Die einzelnen Werte, der dem Cell State neu hinzugefügten Informationen, sind aus dem Intervall [-1;+1], da $\tilde{C}_t$ den

Tangens hyperbolicus, als Aktivierungsfunktion verwendet. Wie sich bereits durch Abbildung 2.17, 2.18 und Abbildung 2.20 andeutet, werden diese Werte dann mit i_t multipliziert. Da it mithilfe der Sigmoid-Funktion berechnet wird, kommt das gleich, ähnlich dem vorigen Schritt, einer Gewichtung, bzw. schematisch einer Entscheidung, welche neuen Informationen, im folgenden dem Cell State hinzugefügt werden sollen. (Der Input in beide Aktivierungsfunktionen ist der selbe, wie der Input, für f_t.) Dieser Prozess des Hinzufügen, wird durch die in Abbildung verdeutlichte Addition unterstrichen. Dieser Teil des LSTMs wird auch als Input Gate bezeichnet.

Zusammenfassend geht also hervor, dass sich der Cell State im aktuellen Schritt durch die Summe von:

- dem Cell-State($\tilde{C}_{t-1}$) , aus dem vorigen Schritt, gewichtet mit f_t
- und den in $\tilde{C}_t$ zusammengefassten neu, aus dem Input ermittelten Informationen, gewichtet mit i_t zusammensetzt.

Eine weitere Entscheidung, die man treffen muss, ist welche der Daten man in den Output hineinnehmen möchte. In o_t wird in einem Sigmoid-Layer(wieder mit Gewichten, und biases) berechnet, welche der Informationen aus dem Cell-State, mit in den Output h_t mitgenommen werden sollen. Dies entspricht wieder einer Gewichtung der Informationen, wie bereits zuvor beschrieben. Der Cell State selbst wird auch nicht direkt übernommen, sondern wird zuvor noch in den Tangens Hyperbolicus gegeben. Der Output aus dem Tangens Hyperbolicus, des Cell States, multipliziert mit o_t ist dann h_t. Insgesamt geht hervor, dass h_t sowohl der Output, als auch der Hidden State ist(siehe Abbildung 2.21).

2.5.4 Gated Recurrent Units

Eine Variation von LSTMs sind Gated Recurrent Units(Abk. GRUs), eingeführt von Cho u. a. (2014). Diese neuronale Netzwerkarchitektur kombiniert die zuvor beschriebenen Gates(das Forget Gate, und das Input Gate), zu eine Update Gate. Des weiteren wird der Cell State und der Hidden State, in dieser Architektur kombiniert.Olah (2015)

Das Update Gate wird in wird wie in Abbildung 2.22, im ersten Schritt, durch das z_t benannte Sigmoid-Layer realisiert. Welches analog zu LSTMs den Hidden State des vorherigen Schrittes, angehängt an den Input des aktuellen Schrittes(x_t), als Input für das Layer nimmt.Olah (2015) In Abbildung 2.22 wurden keine biases für die Layer z_t, r_t und $\tilde{h}_t$ angegeben, in der Realität werden aber auch bei GRU biases in den Layern genutzt. Cho u. a. (2014)

Obwohl die graphische Darstellung Figure 2.22, auf den ersten Blick möglicherweise nicht simpler aussieht, als die Darstellung in Figure 2.16, von LSTMs, so sind doch weniger Berechnungsschritte, mit dieser Architektur nötig, ohne auf die Umsetzung, eines Mechanismus, welcher gleich gut mit weit entfernten Abhängigkeiten, und näher Abhängigkeiten, zurechtkommen kann(i.e. weniger aufwändiges Training, mit ähnlicher Funktionalität). Das sind mögliche Gründe für die hohe Popularität von GRUs im Machine Learning.Olah (2015)

2.5.5 Teacher Forcing

Neuronale Netzwerkarchitekturen, bei welchen der schrittabhängige Output in irgendeiner Form, als Input des nächsten Schrittes dient, können mit einer Methode, namens Teacher Forcing trainiert werden.(Goodfellow u. a., 2016, Seite 377)

Ein Problem, welches sich durch das Nutzen des generellen schrittabhängigen Outputs, als Input für den Hidden State, des nachfolgenden Schrittes ergibt ist, dass der Output nur darauf trainiert wird besonders gut den Trainingsdaten zu gleichen, nicht aber schrittübergreifende Abhängigkeiten besonders gut abzubilden. Natürlich ließe sich das theoretisch, durch semantisch speziell konstruierten Output, mitigieren. Wenn man an ein Natural language processing Problem denkt, wie das generieren von Sätzen, aus mehreren Wort-Atomen, dann enthält ja jedes Wort erstmal keine Informationen, über irgendwelche vorangehenden Abhängigkeiten.(Goodfellow u. a., 2016, Seite 378)

Die Idee beim Teacher Forcing ist es, wie in Abbildung 2.23 angedeutet, beim Training, den tatsächlichen, und damit gleichermaßen auch den erwünschten Output, aus dem vorigen Schritt, als Input für den Hidden State im aktuellen Schritt zu nehmen. Das führt dazu, dass die Schritte im Training komplett unabhängig voneinander ablaufen können, und parallelisiert werden können, da die Loss-Funktion, jeweils nur im Bezug auf die Gewichte und Biases eines Schrittes abgeleitet werden muss. In der tatsächlichen Benutzung des RNNs, nach dem Training, wird dann allerdings wieder der tatsächliche Output, als Input für den aktuellen Schritt genommen.(Goodfellow u. a., 2016, Seite 376 bis 378)

Das Teacher Forcing führt dazu, dass das folgende Log-Likelihood-Funktion maximiert wird. Beispielhaft für die Generierung einer Sequenz der Länge 2.:

$$log\, p\left(y^{(1)},\ y^{(2)} \mid x^{(1)}, x^{(2)}\right) \tag{2.34a}$$

$$= log\ p\ \left(y^{(2)} \mid y^{(1)},\ x^{(1)}, x^{(2)}\right) + log\ p\left(y^{(1)} \mid x^{(1)}, x^{(2)}\right) \tag{2.34b}$$

(Goodfellow u. a., 2016, Seite 378)

Wobei x^t, für den Input in Schritt t steht, und y^t für den Output in Schritt t steht.

Ein Nachteil von strengem Teacher-Forcing ist, dass das so trainierte RNN, nach dem Training mit Outputs konfrontiert ist, die stark von den im Training präsentierten Beispielen abweichen. Dann könnte das mit strengem Teacher Forcing trainierte RNN, also möglicherweise deutlich underperformen. Dies lässt sich verbessern, indem man anteilig den vom RNN produzierten Output, und anteilig den erwünschten Output, als Input für den Hidden State des folgenden Schrittes nimmt. Den Anteil des tatsächlich produzierten Inputs kann man dazu auch fortschreitend im Training langsam erhöhen.(Goodfellow u. a., 2016, Seite 378)

2.6 Graph Neural Networks(GNNs)

Graphen sind Datenstrukturen welchen aus Knoten(engl. Nodes) und Kanten(engl. Edges) bestehen. Graph Neural Networks ist eine Sammelbezeichnung von neuronalen Netzwerkenarchitekturen, welche auf Graphen agieren. Zhou u. a. (2020) Die grundlegende Eigenschaft, die Graph Neural Networks zugrunde liegt ist, dass es auf *Message Passing* zwischen Knoten basiert. Dabei werden Vektor-Nachrichten zwischen den Knoten ausgetauscht, und die Knoten mithilfe von neuronalen Netzwerken geupdatet. (Hamilton, 2020, Seite 48)

Im Folgenden wird dargestellt, wie grundlegende Graph neuronale Netzwerke, welche auch als *Message Passing Neural Networks* bezeichnet werden, generell aufgebaut sind. (Hamilton, 2020, Seite 48)

Als Input erwarten wir einen Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, welcher aus einer Menge an Knoten $\mathcal{V}$, und einer Menge an Kanten $\mathcal{E}$ besteht. Des weiteren erwarten wir, dass die Knoten eine Reihe an Eigenschaften haben: $\mathbf{X} \in \mathbb{R}^{d \times |\mathcal{V}|}$.(Die Eigenschaften sollen die Dimensionalität d haben, und mit reellen Zahlen befüllt sein. Des weiteren soll ein solcher Eigenschaftsvektor für jeden Knoten exisitieren.) (Hamilton, 2020, Seite 48)

Während jeder Iteration die das GNN durchläuft, wird ein Hidden Embedding $\mathbf{h}_u^{(k)}$, für jeden Knoten $u \in \mathcal{V}$ generiert. Dieses Hidden Embedding basiert auf der Nachbarschaft $\mathcal{N}(u)$ des Knotens u.(Hamilton, 2020, Seite 48 - 49)

Diese Knoten-Update kann wie folgt dargestellt werden:

$$\mathbf{h}_u^{(k+1)} = \textsc{update}^{(k)}(\mathbf{h}_u^{(k)}, \textsc{aggregate}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})) \tag{2.35}$$

Oder anders formuliert:

$$\mathbf{h}_u^{(k+1)} = \textsc{update}^{(k)}(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}_{(u)}}^{(k)}) \tag{2.36}$$

(Hamilton, 2020, Seite 49)

Dabei sind UPDATE und AGGREGATE neuronale Netzwerke. $\mathbf{m}_{\mathcal{N}(u)}$ stellt die Nachricht(engl. message) dar, die durch die Aggregation der Nachbarschaft $\mathcal{N}(u)$ des Knotens u, zusammengesetzt ist. Die hochgestellten Buchstaben sollen darstellen, in welcher Iteration/welchem Schritt man sich gerade befindet. In jeder Iteration k, des GNNs nimmt die AGGREGATE-Funktion, sich die Menge an Knoten-Embeddings, in der Nachbarschaft des betrachteten Knoten, und generiert die besagt Nachricht $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$, aus den Nachbarschaftsinformationen(wie in Abbildung 2.24 schematisch dargestellt). Die UPDATE-Funktion kombiniert dann die Nachricht $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$ mit dem eigenen Embedding der vorigen Iteration $\mathbf{h}_u^{(k-1)}$. (Die verschiedenen Iterationen werden manchmal auch als die verschiedenen „Layer“ eines GNNs bezeichnet.) Die initialen Embeddings für die Knoten sind die Eigenschaften, die über die Knoten, im Input enthalten sind: $\mathbf{h}_u^{(0)} = \mathbf{x}_u, \forall u \in \mathcal{V}$. (Hamilton, 2020, Seite 49)

Nachdem das GNN *Message Passing* für eine gesetzte Anzahl K Iterationen gelaufen ist, kann man den Output $\mathbf{z}_u$ für jeden Knoten u wie folgt definieren:

$$\mathbf{z}_u = \mathbf{h}_u^{(K)}, \forall u \in \mathcal{V} \tag{2.37}$$

(Hamilton, 2020, Seite 49)

Nach einer gewissen Anzahl von Iterationen enthalten die Node Embeddings strukturelle Informationen, über den Graphen $\mathcal{G}$, die sie als Input bekommen haben. Des weiteren lernen sie

die Eigenschaften der Knoten um sich herum. (Hamilton, 2020, Seite 50)

Das hier dargestellte Graph neuronale Netzwerk soll nur einen sehr grundlegenden Überblick bieten. Es existieren sehr viele verschiedene Graph neuronale Netzwerkarchitekturen und -konzepte, die weit über das hier vorgestellte hinaus gehen. Zhou u. a. (2020)

2.6.1 Gated Graph Neural Networks(GGNNs)

Gated Graph Neural Networks(GGNNs) sind eine Form von Graph neuronalen Netzwerken, welche in der Lage sind nicht sequenziellen Output zu generieren. Architekturell ist die größte Änderung zu grundlegenden GNNs, dass intern eine Gated Recurrent Unit, wie aus Cho u. a. (2014), enthalten. Das rekurrente GNN, wird dann für eine festgesetzte Anzahl von T Schritten genutzt. Li u. a. (2016)

Propagation Model Die grundlegende rekurrente Struktur von GGNNs ist wie folgt definiert:

$$\mathbf{h}_v^{(1)} = \mathbf{x}_v^\top \tag{2.38}$$

$$\mathbf{a}_v^{(t)} = \mathbf{A}_{v:}^\top \left[\mathbf{h}_1^{(t-1)\top} \dots \mathbf{h}_{|\mathcal{V}|}^{(t-1)\top}\right]^\top + \mathbf{b} \tag{2.39}$$

$$\mathbf{z}_v^t = \sigma(\mathbf{W}^z \mathbf{a}_v^{(t)} + \mathbf{U}^z \mathbf{h}_v^{(t-1)}) \tag{2.40}$$

$$\mathbf{r}_v^t = \sigma(\mathbf{W}^r \mathbf{a}_v^{(t)} + \mathbf{U}^r \mathbf{h}_v^{(t-1)}) \tag{2.41}$$

$$\widetilde{\mathbf{h}_v^{(t)}} = \tanh(\mathbf{W}\mathbf{a}_v^{(t)} + \mathbf{U}(\mathbf{r}_v^t \odot \mathbf{h}_v^{(t-1)})) \tag{2.42}$$

$$\mathbf{h}_v^{(t)} = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{(t-1)} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}_v^{(t)}} \tag{2.43}$$

Li u. a. (2016)

- Dabei steht $\mathbf{h}_v^{(t)}$ für den schrittabhängigen Hidden State eines Knotens.
- $\mathbf{x}_v$ steht für den Feature-Vektor eines Knotens.

- $\mathbf{A}_v \in \mathbb{R}^{|\mathcal{V}|}$ ist ein Vektor, welcher die Knoten, mit denen v verbunden ist kennzeichnet.(Auch dieser Teil ist leicht abgeändert zu Li u. a. (2016), da wir ein ungerichteten Graphen betrachten wollen, und das Paper GGNNs anhand eines Beispiels mit gerichteten Graphen vorstellt.)
- $\mathbf{a}_v^{(t)}$ kann man hier als Aggregationsfunktion verstehen. Diese wird schrittabhängig vorgenommen.
- Deutlich wird auch, dass die Formeln 2.40 bis 2.43 die Formeln sind, die den GRU-Teil der GGNN-Architektur darstellen. So sind z und r, die Update und Reset-Gates.
- σ steht in diesem Kontext für die Sigmoid-Funktion.
- Hier steht $\top$ für eine Transposition.
- (Die in Formel 2.38 ist eine leichte Abwandlung der Formel, die in dem Paper Li u. a. (2016) vorgestellt wurde, da diese sehr konkret auf eine für diese Arbeit nicht interessante Problemstellung zugeschnitten wurde.)

Li u. a. (2016)

Output Models Es existieren verschiedene Möglichkeiten den Output von GNNs zu nutzen. Möchte man einen Knoten auswählen, als Output für eine Problemstellung, kann man $o_v = g(\mathbf{h}_v^{(T)}, \mathbf{x}_v)$ für jeden Knoten $v \in \mathcal{V}$ berechnen, und dann eine Softmax-Funktion auf alle Knoten-Outputs anwenden. Li u. a. (2016)

Benötigt man Graph-Level-Outputs kann man diese zum Beispiel wie folgt erhalten: Li u. a. (2016)

$$\mathbf{h}_{\mathcal{G}} = \tanh\left(\sum_{v\in\mathcal{V}} \sigma\left(i(\mathbf{h}_v^{(T)}, \mathbf{x}_v)\right) \odot \tanh\left(j(\mathbf{h}_v^{(T)}, \mathbf{x}_v)\right)\right) \tag{2.44}$$

Li u. a. (2016)

Dabei agiert der Term $\sigma(i(\mathbf{h}_v^{(T)}, \mathbf{x}_v))$, als Gewichtungsmechanismus für die Informationen, welche für die Graphbezogene-Aufgabe gerade wichtig sind.(g, h und i notieren hier neuronale Netzwerke.) Li u. a. (2016)

2.7 1D-Convolutional Neural Networks(1D-CNNs)

2.7.1 Convolutionale neuronale Netzwerke generell

Convolutionale neuronale Netzwerke(CNNs) sind mehrere Layer an Convolutionen, auf deren Outputs nichtlineare Aktivierungsfunktionen wie tanh oder ReLU angewendet werden. Jedes convolutionale Layer wendet eine Vielzahl von Filtern auf die erhaltenen Daten an, und kombiniert diese dann, um einen Output für ein Layer zu generieren. Die Filter werden mit zufälligen Gewichten initialisiert, und dann beim Training des Convolutionalen neuronalen Netzwerks gelernt. Zwei grundlegend wichtige Eigenschaften von convolutionalen neuronalen Netzwerke sind Positions-Invarianz und Kompositionalität. Positions-Invarianz beschreibt den Fakt, dass es egal ist wo in einem Inputdateneintrag, eine Eigenschaft enthalten ist. Das CNN wird diese unabhängig von der Lage lernen können. Jeder Filter kombiniert einen lokalen Teil von den Inputinformationen(„lower level"-Eigenschaften) zu einer „higher level"-Repräsentation(Kompositionalität). Britz (2015)

2.7.2 1D-Convolutions

Eine eindimensionale Convolution ist eine Operation zwischen einem Vektor von Gewichten $\mathbf{m} \in \mathbb{R}^m$, und einem Inputvektor, welcher durch eine Sequenz $\mathbf{s} \in \mathbb{R}^s$ dargestellt wird. Der Vektor $\mathbf{m}$ ist dabei der Filter der Convolution. Man kann über die Input-Sequenz als Satz betrachen, und $\mathbf{s}_i \in \mathbb{R}$ als eine einzelnes Merkmal, welches mit dem i-ten Wort aus dem Satz assoziiert ist. In einer eindimensionalen Convolution nimmt man das elementweise Produkt des Filters mit jedem m-gram(eine Sequenz aus m Elementen), welches in der Input-Sequenz enthalten ist. Das Ergebnis dieser Operation ist ein anderer Satz c: Kalchbrenner u. a. (2014)

$$\mathbf{c}_j = \mathbf{m}^\top \mathbf{s}_{j-m+1:j} \tag{2.45}$$

(Die einzelnen Stellen im Output werden mit j indexiert.)
Kalchbrenner u. a. (2014)

2.7.3 1D-Convolutionale neuronale Netzwerke für die Verarbeitung von Zeichenketten

Die atomaren Bestandteile der Inputdaten mit denen man in der Regel im *Natural Language Processing* arbeitet, sind Wörter, oder einzelne Zeichen. Um diese in eine neuronale Netzwerkwerkarchitektur zu füttern, generiert man niedrigdimensionale Repräsentationen des Inputs(*Word Embeddings*), oder man nutzt einfache One-Hot-Encodings des Inputs. Hat man

jetzt beispielsweise einen 7 Worte langen Satz, und ein 5-dimensionales Embedding, so wird der Input ingesamt 7×5 groß sein. Britz (2015)

2.7.4 CNN Hyperparameters

Schmale vs. breite Convolutionen Es gibt zwei verschiedene mögliche Convolutionen, welche sich aus Formel 2.45 ergeben. Der erste Typ ist eine schmale Convolution(engl. narrow convolution). Für diese muss $s \geq m$ gegeben sein. Als Ergebnis des schmalen Convolutionstyps ist $c \in \mathbb{R}^{s-m+1}$, mit j zwischen m und s. Der zweite ist eine breite Convolution, für welchen keine Vorbedingungen für s oder m gegeben sein müssen. Das Ergebnis einer breiten Convolution ist: $c \in \mathbb{R}^{s+m-1}$. Dabei ist j zwischen 1 und $s + m - 1$. Kalchbrenner u. a. (2014) Konzeptionell bedeutet das, dass man in Filter die aus dem Input herauslaufen, aber zum Teil auf Inputdaten liegen, die Stellen, die nicht über Werten aus dem Input liegen mit Nullen auffüllt, bei breiten Convolutionen. Dies bezeichnet man als *Zero Padding*. Britz (2015)

Pooling Layers Ein Schlüsselaspekt von CNNs sind Pooling Layers, welche typischerweise nach Convolutions-Layern angewandt werden. Die am häufigsten genutze Pooling Operation ist, dass man von jedem Ergebnis, jeder Filter-Operation, den Maximalwert nimmt(das sogenannte *Max-Pooling*). Britz (2015)

Ein vorteilhafte Eigenschaft von Pooling ist, dass Pooling genutzt werden kann um Output einer festgesetzten Größe zu erhalten. Wenn man beispielhaft mit 1000 Filtern arbeitet, auf welche man Max-Pooling anwendet, erhält man immer 1000-dimensionalen Output. Das erlaubt das Arbeiten mit variabel großen Inputs. Britz (2015)

Ein weiterer Vorteil von Pooling ist es, dass man die Output-Dimensionalität verringert. Im besten Fall bleiben bei diesem Schritt die wichtigsten Informationen erhalten. Man kann sich in diesem Zusammenhang vorstellen, dass jeder Filter nach einer speziellen Eigenschaft in den Inputdaten sucht. Findet er eine diese Eigenschaft zum Beispiel an einer Stelle in einer Inputsequenz, könnten die Output-Werte an dieser stelle besonders hoch sein, und für den Rest der Inputsequenz eher niedrig. Wendet man jetzt das Max-Pooling auf die Filter an, bleibt nur die wichtigste Information erhalten, ist diese Eigenschaft in dieser Instanz der Inputdaten enthalten, und in welcher relativen Region der Inputdaten ist diese Eigenschaft enthalten. Britz (2015)

Stride Size Der Stride, oder die Stride Size(deutsch. Schrittweite), sagt wie weit der Filter nach jeder Anwendung weiter verschoben wird. Kleinere Stride Sizes führen dazu, dass Filter-Anwendungen überlappen. Größere Stride-Sizes führen zu kleinerem Output, und kleinere Stride Sizes zu größerem Output. Das Beispiel in Abbildung 2.28 zeigt ein Beispiel mit zwei verschiedenen Stride Sizes. Eine Stride Size von 1 wird oft genutzt in der wissenschaftlichen Literatur. Mit größeren Stride Sizes lassen sich Models bauen welche ähnliche Eigenschaften haben, wie rekurrente neuronale Netzwerke. Britz (2015)

Channels Channels, oder deutsch Kanäle stellen verschiedene Sichweisen auf die Daten dar, ähnlich wie weitere Dimensionen. Diese Betrachtungsweise kommt aus der Bildverabeitung, wo es beispielweise bei Bildern RGB(rot, grün und blau), als Kanäle gibt. Man wendet die gleichen Convolutionen auf die verschiedenen Kanäle an, entweder mit gleichen oder verschiedenen Gewichten. In der Verabeitung von Sequenzdaten könnt man für die verschiedenen Kanäle beispielsweise verschiedene Embeddings nutzen. Britz (2015)

2.8 Die betrachteten variationalen Autoencoder Architekturen/Konzepte

2.8.1 ChemVAE

ChemVAE, so wird der in dem PaperGómez-Bombarelli u. a. (2018)(mit dem zugehörigen Github-RepositoryWei u. a. (2021)) beschriebene Autoencoder im folgenden genannt. Das Paper beschreibt einen variationalen Autoencoder zum konvertieren diskreter Molekülrepräsentationen, in multidimensionale kontinuierliche Molekülrepräsentationen. Er unterscheidet sich im Vorgehen, und der Modell-Architektur, zu anderen Ansätzen wie folgt:

Die genutzte VAE-Architektur besteht aus:

- Dem Encoder: bestehend aus drei 1D-CNN Layern, mit Filter-Größen: 9, 9 und 10; und 9, 9, 11 Konvolutionskernen. Gefolgt von einem fully-connected Layer, bestehend aus 196 Neuronen.
- Einem Property-Predictor(i.e. einer Struktur zum Vorhersagen, von Eigenschaften der Moleküle): bestehend aus zwei fully-connected Layers, aus jeweils 1000 Neuronen. Es handelt sich beim Property-Predictor also um ein Multilayer-Perceptron, welches die Eigenschaften von Molekülen, anhand ihrer kontinuierlichen Darstellungen vorhersagen

soll. Dieser Teil wurde mit einem Dropout von 0.2 trainiert. Die Objectives(Eigenschaften), auf welche der Property-Predictor trainiert wurde, sind logP, QED und SAS.* In dem Scope dieser Arbeit wird dieser Teil der Architektur allerdings nicht genauer untersucht.

- Dem Decoder: Bestehend aus 3 Layern eines Gated Recurrent Unit(GRU) Netzwerks, mit einer hidden Dimension der Größe 488. Das Output-GRU-Layer hat einen zusätzlichen Output, der das eine Schritt(time-step) vorher gesamplete Zeichen, aus einer Softmax-Verteilung, darstellt. Dieser Teil wurde mithilfe von Teacher-forcing trainiert.**

* Diese Eigenschaften, wurden für den Zinc-Datensatz so gewählt, allerdings sind beliebige Moleküleigenschaften an dieser Stelle denkbar.

** Das Teacher-Forcing führt zu einer erhöhten Accuracy der generierten SMILES-Strings, was auch den Anteil der validen generierten SMILES-Strings, welche aus zufällig gesampelten, nicht in den Trainingsdaten enthaltenen Punkten aus dem Latent Space erhöht, gleichzeitig aber auch das Training insofern erschwert, als dass der Decoder dann tendenziell, das variationale Encoding(den Punkt im latent Space) ignoriert, und sich nur noch auf die tatsächliche Input-Sequenz konzentriert.

Gómez-Bombarelli u. a. (2018)

ChemVAEGómez-Bombarelli u. a. (2018) nutzte eine Untermenge von 35 verschiedenen Zeichen, um den ZINC-DatensatzWei u. a. (2021), als SMILES dargestellt, zu encoden. Die Länge der als SMILES dargestellten Moleküle wurde auf 120 Zeichen begrenzt. Gómez-Bombarelli u. a. (2018) Kürzere Molekül-Strings wurden mit Leerzeichen auf 120 aufgefüllt. Die Molekül-Stringzeichen werden einzeln in one-hot-Array-Encodings umgewandelt, und konkateniert. Wei u. a. (2021) Diese Begrenzung stellt keine inhärente Limitation des Ansatzes dar, wurde aber gewählt, um die Berechnung zu erleichtern. Es wurden nur kanonisierte(normierte) Darstellungen von SMILES, zum trainieren des VAEs, genutzt, um nicht mit verschiedenen Repräsentationen des gleichen Moleküls händeln zum müssen. Keras Chollet u. a. (2015) und Tensorflow citetensorflow2015-whitepaper wurden zum initialisieren und trainieren des Modells genutzt, und das *Rdkit*-package Landrum u. a. (2020) für die chemoinformatischen Aspekte des Projektes. Einige der Angaben gelten nur für den ZINC-Datensatz, im selbigen Paper wurde zusätzlich noch der QM9-DatensatzRamakrishnan u. a. (2014) beleuchtet. Einige der Hyperparameter wurden dafür anders gewählt. Dies liegt vermutlich an dem Umstand, dass die in diesem Datensatz enthaltenen Moleküle, im Durchschnitt andere Eigenschaften haben. Gómez-Bombarelli u. a. (2018) Da das aber für den Scope meiner Bache-

lorarbeit nicht im Detail wichtig ist, werde ich darauf im folgenden nicht genauer eingehen.

Der VAE und der Property-Predictor wurden gemeinsam trainiert. Der Regressionsloss(mittlere quadratische Abweichung/mean squared error(abbr.: mse)), des Property-Predictors wurde in die Loss Funktion des variationalen Autoencoders mit einbezogen. Der Gesamt-Loss des Models besteht also aus dem Rekonstruktionsloss(kategorische Cross-Entropie), dem KL-Loss, und dem Regressionsloss des Property-Predictors. Diese Loss Terme sind jeweils gewichtet.Gómez-Bombarelli u. a. (2018) (Wei u. a., 2021, in ./chemvae/train_vae.py und ./chemvae/hyperparameters.py)

Die kategorische Cross-Entropie: $E_{CCE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}(p_{ic}\, log(y_{ic}))$

- Der betrachtete Teil der Trainingsdaten besthet dabei aus N-Datenpaaren(z.B. eine Batch) $\{(x_1,\ t_1), (x_2,\ t_2) \dots (x_N,\ t_N)\}$
- wobei $x \in R^{\gamma}$ den i-ten Input-Vektor, der Länge γ
- $y_i \in R^C$ beschreibt den C-dimensionalen Output
- $t_i \in R^C$ beschreibt den erwünschten Output, für den Input x_i
- p_{ic} ist eine binäre Indikator Funktion, die erkennt, ob das i-te Trainingspattern zu der c-ten Kategorie gehört.
- p_{ic} kann als der tatsächliche Output interpretiert werden.

Chen u. a. (2020)

KL-Loss = die KL-Divergenz zwischen der generativen Verteilung des VAEs(beispielhaft als $q(z|x)$ notiert), und der tatsächlichen Wahrscheinlichkeitsverteilung $p(z|x)$.

Es wurde insgesamt für 120 Epochen trainiert. Nach 29 Epochen wird der variationale Loss nach dem Sigmoid schedule gewichtet(learning rate annealing). (Annealing(engl. abkühlen/ausglühen), beschreibt in diesem Zusammenhang, bezogen auf die Learning Rate den Umstand, dass man mit einer großen initialen Learning Rate beginnt, und diese dann kleiner wird. Nakkiran (2020)))

$$VAEweight(t) = VAEweight(t-1) * sigmoidaleSchedulingFunktion(t) \quad (2.46)$$

$$sigmoidaleSchedulingFunktion(t) = \frac{1}{1 + e^{slope * (start - t)}} \quad (2.47)$$

(Wei u. a., 2021, in ./chemvae/mol_callbacks.py)

Die slope(Steigung), und der start Parameter, sind in diesem Fall setzbare Hyperparameter, eines Durchgangs. Für alle konkret, gesetzten, genutzten Hyperparameter: siehe exp.json(Wei u. a., 2021, in ./models/zinc_properties/exp.json). Generelle Hyperparameter, teilweise überschrieben, durch Parameter, welche im exp.json explizit gesetzten werden, finden sich in (Wei u. a., 2021, in ./chemvae/hyperparameters.py und ./models/zinc_properties/exp.json).

Aufgrund der fragilen Natur, der SMILES-Repräsentation, und des hier gewählten zeichenweisen Generierungsansatzes, kann es bei diesem Ansatz zur Generierung invalider Molekülstrukturen kommen. Bei der Generierung neuer SMILES-Strings, kam es zu Validitätsraten zwischen 70% und weniger als 1%. Es wurden im Verlauf, des hier behandelten Papers allerdings auch mehr als nur der eine, verhältnismäßig erfolgreiche architekturelle/konzeptionelle Ansatz betrachtet. Vorgestellt, und von mir im Folgenden wird allerdings nur der erfolgreichste Ansatz, inklusive des Property-Predictors behandelt. Es wurde unteranderem mit einem rekurrenten neuronalen Netzwerk, als Encoder, und verschiedenen Hyperaparametern experimentiert. Gómez-Bombarelli u. a. (2018)

Der variationale Autoencoder generiert aber auch manchmal Moleküle, die formal valide sind, als Molekülgraphen aber nicht erwünschten Molekülstrukturen entsprechen, da sie in der Realität nicht besonders stabil sind, oder prohibitiv schwer zu synthetisieren sind. Beispiele für solche Gruppen sind Säurechlorid, Anhydride, Cyclopentadiene, Aziridine, Enamine, Hemiaminale, Enole, Ether, Cyclobutadiene, und Cycloheptatriene. Gómez-Bombarelli u. a. (2018)

Die Autoren des Papers räumen in ihrer Conclusion ein, dass es effizienter wäre, wenn man den Autoencoder dazu bringen würde, nur valide SMILES-Strings zu generieren, geben aber auch zu bedenken, dass der von ihnen gewählte Ansatz leichtgewichtig ist, und dadurch, dass er nicht gezwungen wird, nur valide Strukturen zu lernen, eine größere Flexibilität, bei dem Lernen der Architektur von SMILES hat.Gómez-Bombarelli u. a. (2018)

Der hier vorgestellte variationale Autoencoder ist ein zweifach probabilistisches Modell. Zusätzlich zu dem Noise, was durch den Sampling-Prozess, nach dem Encoding entsteht, sampelt der Decoder zusätzlich einen String aus der Wahrscheinlichkeitsverteilung, der einzelnen Zeichen, an jeder Position, die vom letzten Layer generiert wird. Also ist in diesem Fall der Decoding-Prozess nochmal stochastischer, als das bei einem anderen VAE der Fall wäre.Gómez-

Bombarelli u. a. (2018)

Für die meisten Punkte im latent space wird ein Molekül mit einer hohen Frequenz, aus einem betrachteten Punkt decodet, wenn man den Decoding Prozess mehrfach wiederholt. Andere, ähnliche Moleküle werden mit abfallenden geringeren Frequenzen, aus dem selben Punkt decoded. Gómez-Bombarelli u. a. (2018) Das ist ein Anhaltspunkt dafür, dass der VAE tatsächlich relevante Moleküleigenschaften in seinem latent space abbildet.

Es wurde nach dem Training der VAE-Architekturen eine PCA(hochdimensional => zweidimensional) des latent space, mit einigen der betrachteten Architekturen gemacht. Dann wurde dieser latent space für einige der Eigenschaften, auf welche der Property-Predictor trainiert wurde, farblich eingefärbt. Ein daraus hervorgehendes erwähnenswertes Faktum ist, dass wenn der Autoencoder, ohne die Property-Prediction trainiert wird, sich der latent space nicht nach einer Eigenschaft ordnet. Mit Property-Predictor trainiert allerdings spaltet sich der latent space, in der zweidimensionalen Darstellung nach PCA, in zwei Extreme, der betrachteten Eigenschaft, und einem weichen Übergang, zwischen diesen beiden.Gómez-Bombarelli u. a. (2018)

2.8.2 Fragment-based VAE

Fragment-based VAE, so wird der in dem PaperPodda u. a. (2020) beschriebene variationale Autoencoder im folgenden genannt. Die generelle Idee des Ansatzes ist es Moleküle in Fragment-Sequenzen zu zerlegen, wobei die Fragmente Wörter darstellen.Podda u. a. (2020)

Ein Problem, was sich dieser Ansatz als Ziel setzt zu lösen sind die niedrigen Validitätsraten, die man bei klassischen Language-Model-basierten generativen Modellen erhält. Diese generieren Moleküle Atom für Atom, oder Zeichen für Zeichen. Dafür nehmen sie Inspirationen aus dem *Fragment based Drug Design*. Das heißt im konkreten, dass sie ein generatives Modell präsentieren, welches Moleküle Fragment für Fragment generiert.Podda u. a. (2020)

Ein zweiter Ansatz, der in diesem Paper vorgestellt wird, ist für den Rahmen dieser Bachelorarbeit nicht geeignet. In diesem schlagen die Wissenschaftler ein anderes Vorgehen zum encoden der Moleküle vor, welches die Raten der einzigartigen(engl. *unique*) Moleküle erhöhen soll. Dazu verwenden sie einen Ansatz, welchen sie selber als *Low Frequency Masking(LFM)* bezeichnen. Die Idee dabei ist es, dass sie Fragmente, die nur mit einer geringen Frequenz vorkommen, mit einem Token, welches ihre Häufigkeit kodiert, maskieren. Bei der Nutzung des generativen

Modells des variationalen Autoencoders, sampeln sie zufällig aus der Verteilung der Moleküle, welche durch dieses Token maskiert wurden.Podda u. a. (2020) Diese zusätzliche Stochastizität bringt vermutlich keinen Vorteil, bei der Richtigkeit(engl. accuracy), und Organisation des Latent Spaces.

Die Autoren räumen selber ein, dass sich dieser LFM-Ansatz möglicherweise nicht für molekulare Optimierungsprobleme eignet, aufgrund des Faktes, das der Latent Space weniger strukturiert ist, als bei anderen Ansätzen.Podda u. a. (2020) Aufgrund dieses Faktes eignet sich dieser Teilansatz nicht für die genauere Betrachtung, im Rahmen dieser Bachelorarbeit.

Um die Moleküle, wie beschrieben, fragmentiert zu en- und decoden, müssen diese vorerst fragmentiert werden. Dazu macht sich dieses Paper den *Breaking of Retrosynthetically Interesting Chemical Substructures(BRICS)*-AlgorithmusDegen u. a. (2008) zunutze. Dieser bricht die Bindungen in den Molekülen nach einer Menge von möglichen chemischen Reaktionen auf. Dann werden an den Enden dieser aufgebrochenen Endungen/dieser Fragment, „Dummy"-Atome hinzugefügt.(Hier durch Sternchen notiert.) Diese „Dummy"-Atome stellen die Stellen des Fragmentes dar, an denen die Fragmente, mit anderen Fragmenten zu einem Molekül verknüpft werden können. Der hier verwendete Ansatz scannt die als SMILES gespeicherten Moleküle von links nach rechts, und bricht die dargestellten Bindungen auf, sobald diese mit einer der in den BRICS-Regeln beschriebenen Reaktionen, übereinstimmt. Das dadurch enstandene linke Fragment wird gespeichert, und das Molekül weiter rekursiv gescannt, bis man am Ende des Moleküls angekommen ist. Dieser Prozess ist insgesamt reversibel. Das bedeutet, man kann ein Molekül eindeutig wieder zusammenführen, wenn man die Sequenz der zugrundeliegenden Fragmente vorliegen hat.Podda u. a. (2020)

Die Moleküle liegen jetzt als Sequenzen SMILES-notierter Fragmente vor. Schematisch wereden die Moleküle, als Sätze angesehen, und die Menge der zugrunde liegenden Fragmente wird als Vokabular, von Wörtern angesehen. Die Fragmente werden embeddet. Dabei werden Fragmente, die in einem ähnlichen Kontext auftreten, in einander nahen Regionen im Embedding Space. Formal ausgedrückt bedeutet das:

- Gegeben einer Sequenz $s = (s_1, s_2, \ldots, s_{|s|})$, wird die folgende Loss-Funktion minimiert:

$$\mathcal{L}(s) = -\sum_{i=1}^{|s|} \sum_{-w \leq j \leq w} log\ P(s_{i+j}|s_i),\ j \neq 0 \tag{2.48}$$

- Dabei ist w die Größe des Kontext-Fensters.
- Das Zeichen s_i steht für das Ziel Fragment.
- P ist in diesem Paper ein Skipgram Modell, mit Negativ-Sampling. Le und Mikolov (2014)
- Nach dem Training werden die Embeddings, als $x = (x_1, x_2, \ldots, x_{|x|})$ repräsentiert.

Architekturell liegt diesem Modell auch ein variationalen Autoencoder zugrunde.

Podda u. a. (2020)

Der Encoder: Als Architektur, um die Embeddings zu Encoden werden Gated Recurrent Units(GRUs)Cho u. a. (2014) genutzt. Die Embeddings werden also für eine GRU genutzt. Der Encoder wird darauf trainiert die Kullback-Leibler-Divergenz zu minimieren:

$$\mathcal{L}_{enc}(x) \;=\; -KL\left(\mathcal{N}\left(\mu, diag\left(\sigma^2\right)\right) \;||\, \mathcal{N}(0, \mathbb{I})\right)) \tag{2.49}$$

In diesem Paper sind μ und σ wie folgt definiert:

- $\mu = W_\mu h + b_\mu$
- $log(\sigma^2) = W_\sigma h + b_\sigma$
- Wobei W für Gewichtsmatrizen, und b für Biases steht.

Podda u. a. (2020)

Der Decoder: Bei dem hier verwendeten Decoder handelt es sich um ein rekurrentes Model, welches GRUs enthält. Der Hidden State dieser wird mithilfe des Reparametrization TricksKingma und Welling (2014) initialisiert: $z = h_0 = \mu + \sigma\epsilon$ mit $\epsilon \sim \mathcal{N}(0, \mathbb{I})$. Der Decoder berechnet die Output-Wahrscheinlichkeit, des nächsten Elements der Sequenz wie folgt:

$$P(x_{i+1}|x_i, h_{i-1}) = softmax(W_{out}h_i + b_{out}) \tag{2.50}$$

Wobei der Hidden State mithilfe einer GRU berechnet wird: $h_i = \text{GRU}(x_i, h_{i-1})$.

Während des Trainings wird Teacher Forcing genutzt. Der Decoder wird darauf trainiert die negative log-Wahrscheinlichkeit der Fragment-Sequenz zu lernen:

$$\mathcal{L}_{dec}(x) = -\sum_{i=1}^{|x|} logP(x_{i+1}|x_i, h_{i-1}) \tag{2.51}$$

Dieser Loss korrespondiert mit der Cross-Entropie zwischen der One-hot encodeten tatsächlichen Sequenz und den vom Decoder vorhergesagten Fragment-Wahrscheinlichkeiten, für jedes Element der Sequenz.

Podda u. a. (2020)

Der Model-Loss: Dieses Language Model wird auf einem Datensatz von Fragment Sequenzen $\mathcal{D}$ trainiert. Die zusammengefasste Loss-Formel, für das gesamte Model ist wie folgt:

$$\mathcal{L}(\mathcal{D}) = \sum_{x \in \mathcal{D}} \mathcal{L}_{enc}(x) + \mathcal{L}_{dec}(x) \tag{2.52}$$

Podda u. a. (2020)

Der generative Prozess fängt an mit dem Sampeln eines Punktes im Latent Space, welcher durch einen multidimensionalen Vektor dargestellt wird: $z \sim \mathcal{N}(0, \mathbb{I})$. Der erste Input des Decoders ist ein <***SOS***>-Token(SOS steht in diesem Kontext für start of sequence). Tokens und der rekurrente Hidden State, werden durch eine GRU geleitet, und dann in ein lineares + Softmax-Layer, um eine Wahrscheinlichkeitsverteilung, für das nächste Fragment, zu generieren. Das wahrscheinlichste wird dann als das nächste Fragment genommen. Die generative Prozess ist beendet, sobald ein <***EOS***>-Token(EOS = end of sequence) gesampelt wird. Die resultierende Fragment-Sequenz wird dann wieder in ein Molekül umgewandelt. Wichtig dafür, dass das passieren kann ist, dass das erste und das letzte Fragment genau einen Verbindungspunkt(* = Dummy-Atom) haben, und alle Fragmente dazwischen genau zwei Verbindungspunkte. Sequenzen, die diese Bedingung nicht erfüllen, werden abgelehnt.

Im Training wurden zwei Trainingsdatensätze betrachtet. Einmal der ZINC-Datensatz, und einmal der PCBA-Datensatz.

Während des Preprocessings wurden alle Moleküle entfernt, welche aus weniger als zwei Fragmenten bestehen. Das resultierte in 227946 validen Molekülen, aus dem ZINC-Datensatz.

Podda u. a. (2020)

Hyperparameter: Als Embedding Dimension wurde 64 gewählt; Die Anzahl an rekurrenten Layers zu 2; die Anzahl an Einheitem, aus denen die GRUs bestehen sollen zu 128; und die Dimensionlität des Latent Space zu 100. Der Adam Optimizer wurde eingesetzt, mit einer initialen Lernrate von 0.00001 annealed mit einem Multiplipaktionsfaktor von 0.9. Die Batch Size wurde auf 128 gesetzt. Des weiteren wurde eine Dropout-Rate von 0.3 auf die rekurrenten Layer angewandt. Es wurde nur für 4 Epochs(für den nicht LFM-Ansatz) trainiert, da empirisch ermittelt wurde, dass das Model nach 4 Epochs bereits begann zu overfitten.

Podda u. a. (2020)

2.8.3 Constrained Graph VAE(CGVAE)

Ein spezielles variationales Autoencoder-Konzept, was unter anderem aus Gated Graph Neural Networks besteht, wird in Liu u. a. (2018) beschrieben, und wird im folgenden mit CGVAE abgekürzt.

Sowohl der Encoder, als auch der Decoder sind in diesem Projekt mithilfe von Gated GraphNNs umgesetzt, und ein paar zusätzlichen neuronalen Netzwerken realisiert. Der vorgeschlagene Decoder generiert aus dem Latent Space sequentiell einen Graphen. Der in diesem Paper beschriebene Ansatz ist ein Ansatz der generell zum encoden eines Datensatzes ungerichteter Graphen ausgelegt ist, er wird aber in selbigem Paper auch an organischen Molekülen, im SMILES-Format ausprobiert.Liu u. a. (2018)

Encoder

Der Encoder des VAEs ist ein GGNN G_{enc} welcher jeden Knoten im Input Graphen $\mathcal{G}$ in eine diagonale Normalverteilung im d-dimensionalen embeddet. Der Latent-Vektor $\mathbf{z}_v$ wird dann mit den vom Encoder generierten Duchschnittsvektor $\boldsymbol{\mu}_v$ und dem Standardabweichungsvektor $\boldsymbol{\sigma}_v$, als Input, gesampelt. Der Loss des Encoders entspricht dann dem bekannten KL-Loss zwischen der generierten Verteilung und einer Standardnormalverteilung.

Generative Model(Decoder)

Das generative Modell besteht aus mehreren neuronalen Netzwerken. Diese werden mit f, C, L_ℓ und G_{dec} notiert.

Der generative Prozess beginnt mit N Vektoren $\mathbf{z}_v$. Diese stellen konzeptionell die Spezifikation für den zu generierenden Graphen dar. N stellt dabei eine obere Grenze der Anzahl der Knoten, welche im finalen generierten Graphen enthalten sein sollen. Die Generierung der Kanten des Graphen, der darauf folgt, besteht aus den Aktionen `focus` und `expand`. In jedem Schritt wählt die `focus`-Funktion einen Knoten, welcher betrachtet werden soll. Die `expand`-Funktion wählt dann eine mögliche Kante, die ausgehend von dem betrachteten Knoten hinzugefügt werden soll. Die `expand`-Funktion ist nur abhängig von der partiellen Graph-Struktur $\mathcal{G}^(t)$, welche bereits vorliegt, und nicht auf der gesamten Historie an Aktionen, die zu diesem Zustand geführt haben. Liu u. a. (2018)

Knoten - Initialisierung Jeder Knoten v ist mit einem Zustand $\mathbf{h}_v^{(t=0)}$ assoziiert. $\mathbf{h}_v^{(t=0)}$ besteht aus der Konkatenation von $\mathbf{z}_v$ und $\boldsymbol{\tau}_v$(i.e. $\mathbf{h}_v^{(t=0)} = [\mathbf{z}_v, \boldsymbol{\tau}_v]$). $\mathbf{z}_v$ ist ein d-dimensionaler Vektor, welcher aus einer Normalverteilung $\mathcal{N}(0, \mathbb{I})$ gesampelt wurde. $\boldsymbol{\tau}_v]$ ist dabei ein One-Hot Vector, welcher besagt, um welchen Knoten-Typ es sich handelt. $\boldsymbol{\tau}_v]$ erhält man, indem man durch sampeln aus dem Softmax-Output eines gelernten Mappings $\boldsymbol{\tau}_v \sim f(\mathbf{z}_v)$. f ist dabei ein neuronales Netzwerk. Dieser wurde als linearer Classifier implementiert, der den 100-dimensionalen Latent Space, als Input bekommt, und als Output die verschiedenen Knoten-Klassen hat. Liu u. a. (2018)

Mithilfe dieser Knoten-bezogenen Variablen lässt sich eine globale Repräsentation $\mathbf{H}^{(t)}$ berechnen. Diese ist der Durchschnitt der Knotenrepräsentationen, welche im Schritt t bereits zu einem Teilgraphen verbunden sind. Des weiteren lässt sich $\mathbf{H}_{init}$ berechnen. $\mathbf{H}_{init}$ ist der Durchschnitt aller Knotenrepräsentationen zum Zeitpunkt t = 0. Zusätzlich zu den N-Knoten, mit denen begonnen wird, wird ein spezieller Stop-Knoten(⊘) hinzugefügt.

Knoten - Update Jedes Mal, wenn der Graph sich verändert $\mathcal{G}^{(t+1)}$, wird $\mathbf{h}_v^{(t)}$ verworfen, und es werden für alle Knoten neue Repräsentationen $\mathbf{h}_v^{(t+1)}$ berechnet. Dabei wird auch mit einbezogen, wie sich die Nachbarschaftsverhälnisse, zwischen den Knoten möglicherweise verändert haben. Das wird mithilfe eines Gated Graph Neural Networks G_{dec}, als rekurrente Operation für S Schritte umgesetzt.(Hier wird $S = 7$ genutzt.) Liu u. a. (2018)

$$\mathbf{m}_v^{(0)} = \mathbf{h}_v^{(0)} \tag{2.53}$$

$$\mathbf{m}_v^{(s+1)} = \text{GRU}\left[\mathbf{m}_v^{(s)}, \sum_{v \overset{\ell}{\leftrightarrow} u} E_\ell(\mathbf{m}_u^{(s)})\right] \tag{2.54}$$

$$\mathbf{h}_v^{(t+1)} = \mathbf{m}_v^{(S)} \tag{2.55}$$

Liu u. a. (2018)

Die Summe in Formel 2.54 läuft über alle Kanten, des aktuellen Graphen. E_ℓ ist eine Kanten-Spezifisches neuronales Netzwerk. Es wird in diesem Zusammenhang ein Master-Knoten genutzt, wie in Gilmer u. a. (2017) beschrieben.Liu u. a. (2018) Der Master-Knoten ist mit allen anderen Knoten im Netzwerk verbunden. Gilmer u. a. (2017)

Dadurch, dass $\mathbf{h}_v^{(t+1)}$ ausgehend von $\mathbf{h}_v^{(0)}$, und nicht ausgehend von $\mathbf{h}_v^{(t)}$, ist es unabhängig von der Generierungshistorie von $\mathcal{G}^{(t+1)}$. Liu u. a. (2018)

Kanten - Auswahl und Labelling Als erstes wird, wie bereits beschrieben ein Fokus-Knoten v ausgewählt. (Die `focus` Operation ist als Queue umgesetzt, mit zufälligem Initialknoten.) Die `expand`-Funktion sucht sich dann eine Kante $v \overset{\ell}{\leftrightarrow} u$ aus. Die Kante geht vom Knoten v zum Knoten u, und ist mit dem Label ℓ versehen. Für jeden Knoten u, der nicht im aktuellen Schritt fokussiert ist, wird ein Feature-Vektor $\boldsymbol{\phi}_{v,u}^{(t)}$. Dieser ist wie folgt definiert: $\boldsymbol{\phi}_{v,u}^{(t)} = [\mathbf{h}_v^{(t)}, \mathbf{h}_u^{(t)}, d_{v,u}, \mathbf{H}_{init}, \mathbf{H}^{(t)}]$. Dabei ist $d_{v,u}$ die Distanz der zwei Knoten u und v im Graphen. Liu u. a. (2018)

Der Feature-Vektor gibt dem Model, sowohl lokale Informationen, über den Fokus-Knoten u, als auch über die möglichen zukünftigen Kanten($\mathbf{h}_v^{(t)}$,$\mathbf{h}_u^{(t)}$), welche von dem Fokus-Knoten ausgehen. Des weiteren enthält das Model über den Feature-Vektor Informationen über die initiale Graph-Spezifikation $\mathbf{H}_{init}$ und den aktuellen Graph-Zustand $\mathbf{H}_{init}$. Diese Repräsentationen werden genutzt, um eine Verteilung über die potentiellen zukünftigen Kanten zu generieren:Liu u. a. (2018)

$$p(v \overset{\ell}{\leftrightarrow} u|\boldsymbol{\phi}_{v,u}^{(t)}) = p(\ell|\boldsymbol{\phi}_{v,u}^{(t)}, v \leftrightarrow u) \cdot p(v \leftrightarrow u|\boldsymbol{\phi}_{v,u}^{(t)}) \tag{2.56}$$

$$p(v{\leftrightarrow}u|\boldsymbol{\phi}_{v,u}^{(t)}) = \frac{M_{v\leftrightarrow u}^{(t)} \exp[C(\phi_{v,u}^{(t)})]}{\sum_w M_{v\leftrightarrow w}^{(t)} \exp[C(\phi_{v,w}^{(t)})]} \tag{2.57}$$

$$p(\ell|\boldsymbol{\phi}_{v,u}^{(t)}) = \frac{m_{\substack{\ell \\ v\leftrightarrow u}}^{(t)} \exp[L_\ell(\phi_{v,u}^{(t)})]}{\sum_k m_{\substack{k \\ v\leftrightarrow w}}^{(t)} \exp[L_k(\phi_{v,u}^{(t)})]} \tag{2.58}$$

Liu u. a. (2018)

Ein neuronales Netzwerk C berechnet, den Ziel-Knoten u für die zukünfige Kante, in Form eines Softmax-Outputs. Ein weiteres neuronales Netzwerk L_ℓ, mit Softmax-Output, bestimmt dann den Kanten-Typ. Liu u. a. (2018)

Valenzmaskierung Die Valenz der einzelnen Atome, die hier als Knoten modelliert werden, ist eine wichtige Einschränkung für die Generierung valider Moleküle. Während des Generierungsprozesses stellen spezielle Masken M und m sicher, dass die Anzahl der Bindungen an einem Knoten b_v niemals die Valenz(Bindigkeit) b_v^* des Knotens überschreitet. Sollte am Ende des Generierungsprozess der Fall $b_v < b_v^*$ eintreten, werden die fehlenden Bindungen durch Bindungen zu $b_v^* - b_v$ Wasserstoffatomen für den Knoten/das jeweilige Atom hinzugefügt. Diese Maßnahmen stellen sicher, dass die Moleküle, am Ende des Generierungsprozesses, immer syntaktisch valide sind. Des weiteren werden damit auch Kanten-Duplikation und Self-Loops vermieden. Liu u. a. (2018)

Die Maske $M_{v \leftrightarrow u}^{(t)}$ ist wie folgt definiert:

$$\begin{aligned} M_{v \leftrightarrow u}^{(t)} = \mathbf{1}b_v < b_v^* \times \mathbf{1}b_u < b_u^* \times \mathbf{1}(\text{die Kante } v \leftrightarrow u \text{ existiert noch nicht}) \\ \times \quad \mathbf{1}(v \quad \neq \quad u) \quad \times \quad \mathbf{1}(u \text{ ist noch nicht „geschlossen“}) \end{aligned} \tag{2.59}$$

Liu u. a. (2018)

Dabei ist $\mathbf{1}$ eine Indikatorfunktion. Kanten zum Stopp-Knoten sind immer unmaskiert. Liu u. a. (2018)

Die Maske $m_{v \overset{\ell}{\leftrightarrow} u}^{(t)}$ stellt sicher, das beim wählen des Labels einer Kante keine Valenzregeln verletzt werden. In diesem Fall modelliert das Label, welchen Typ Bindung eine Kante darstellt. Die Werte $\ell = 1, 2, 3$ werden genutzt um Einfach-, Doppel-, und Dreifachbindungen zu modellieren. $m_{v \overset{\ell}{\leftrightarrow} u}^{(t)}$ ist wie folgt definiert:

$$m_{v \overset{\ell}{\leftrightarrow} u}^{(t)} = M_{v \leftrightarrow u}^{(t)} \times \mathbf{1}(b_u^* - b_u \leq \ell) \tag{2.60}$$

Termination Es werden Kanten zum Knoten v hinzugefügt, mit der `expand`-Funktion und $\mathcal{G}_{dec}$, bis eine Kante zum Stop-Knoten ausgewählt wird. Dadurch verliert der Knoten v den Fokus-Zustand, und wird „geschlossen“. Das bedeutet es dürfen keine Kanten mehr zu

diesem Knoten hinzugefügt werden. Dies wird durch die Maske M sichergestellt. Dann wird der nächste Knoten aus der `focus`-Queue ausgewählt. Auf diesem Weg wird eine einzelne verbundene Graph-Komponente aufgebaut. Die Kanten-Generierung läuft weiter, bis die Queue leer ist. (Der gesamte Generierungsprozess führt möglicherweise dazu, dass Knoten, nicht mit dem Graphen verbunden werden. Diese werden am Ende des Generierungsprozesses verworfen.)

Die Loss-Funktion

Die Loss-Funktion ingesamt ist:

$$\mathcal{L} = \mathcal{L}_{reconstruction} + \lambda_1 \mathcal{L}_{latent} \tag{2.61}$$

$$\mathcal{L}_{reconstruction} = \sum_{\mathcal{G} \in \mathcal{D}} \log[p(\mathcal{G}|\mathcal{G}^{(0)}) \cdot p(\mathcal{G}^{(0)}|\mathbf{z})] \tag{2.62}$$

$$\mathcal{L}_{latent} = \sum_{v \in \mathcal{G}} \mathrm{KL}(\mathcal{N}(\boldsymbol{\mu}_v, \mathrm{diag}(\boldsymbol{\sigma}_v)^2 || \mathcal{N}(0, \mathbb{I})) \tag{2.63}$$

Liu u. a. (2018)

Die Loss-Formel 2.61 ist eine generelle VAE-Loss-Formel. An der Rekonstruktions-Loss wird nochmal deutlich, dass auf den resultierenden Graphen optimiert wird, aus dem Latent Space, und nicht darauf geachtet wird, auf welchem Weg der generative Prozess dahin kommt. Liu u. a. (2018) (Ein Term wurde aus dem Paper weggelassen, da er sich auf die Optimierung von Graph-Eigenschaften bezieht, und dieser Teil nicht in den Scope dieser Arbeit fällt.)

Weiteres

Die einzelnen architekturellen Entscheidungen wurden durch einige Versuche, in denen man Teile weggelassen, oder leicht anders genutzt hat, auf ihre Sinnhaftigkeit getetest. Des weiteren wurden noch ein paar andere Experimente und Tests vorgenommen, auf welche in dieser Arbeit nicht weiter eingegangen wird.

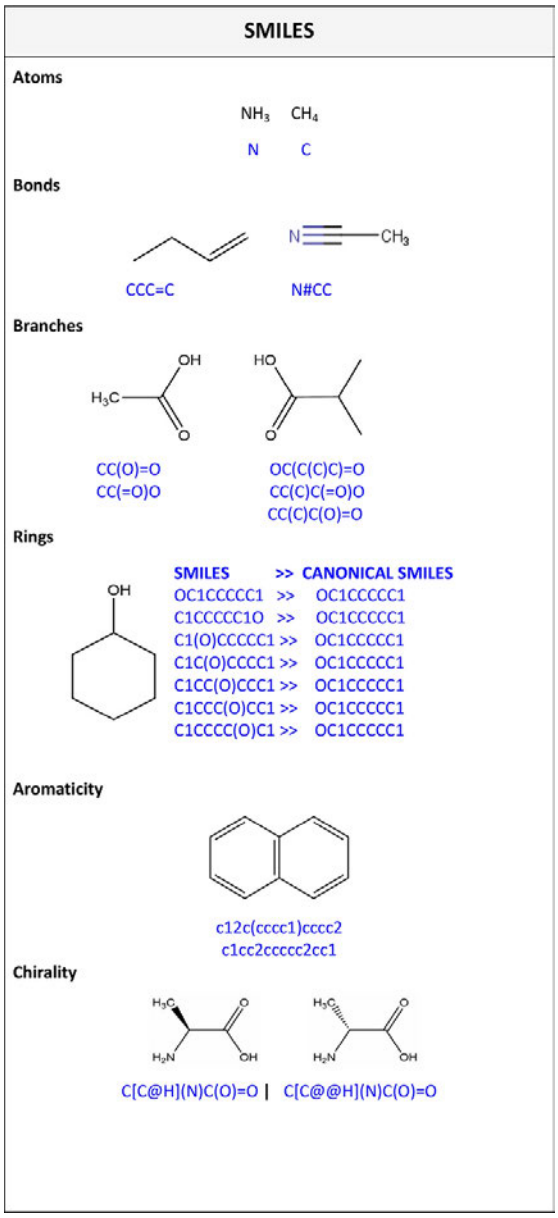


Abbildung 2.3: Eine graphische Darstellung verschiedener organischer Strukturformeln, mit korrespondierenden SMILES(in blau). Unter dem Punkt Atoms(deutsch Atome), werden zwei Beispiele für einzelne Elementsymbol-SMILES dargestellt, deutlich wird auch der Fakt dass die impliziten Wasserstoffatome, nach Valenzregeln, in der SMILES-Darstellung weggelassen werden. Unter dem Punkt Bonds(deutsch Bindungen) werden zwei Beispiele dargestellt. Ein Molekül, welches eine Doppelbindung(=) enthält, und ein Molekül, welches eine Dreifachbindung(#) enthält. Unter dem Punkt Rings werden verschiedenen SMILES-Encodings für ein Cyclohexanol-Molekül dargestellt, zusammen mit der kanonisierten SMILES-Darstellung. Ein Ring wird in SMILES wird durch die selbe Zahl am öffnenden, und schließenden Atom symbolisiert. Weininger (1988) Unter dem Punkt Aromaticity(deutsch Aromatitzität) werden zwei mögliche SMILES-Darstellungen für das selbe aromatische Ringsystem dargestellt. In SMILES werden aromatische Ringe durch Ringe mit kleinen Elmentsymbolen dargestellt. Unter dem Punkt Chirality(deutsch Chiralität) sind zwei verschiedene Konstitutionsisomere(speziell Enantiomere) dargestellt.

Source: Saldívar-González u. a. (2020)

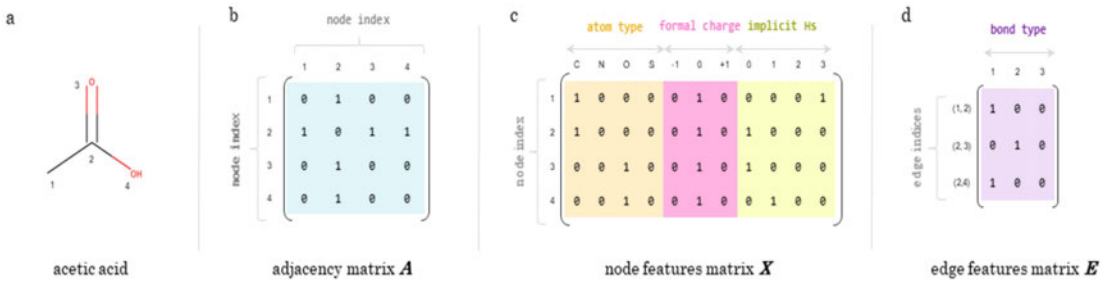
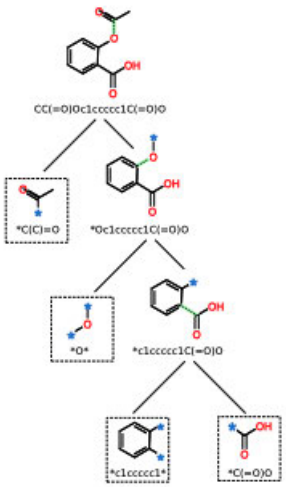


Abbildung 2.4: Eine Darstellung, welche eine Möglichkeit darstellt, ein Molekül, als Graph zu encoden. Das Molekül, welches hier als Beispiel genutzt wird ist das Essigsäure-Molekül(engl. acetic acid). Die Bindungen, werden in einer Adjazenz-Matrix dargestellt. Des weiteren gibt es noch eine Node-Feature-Matrix. Diese enthält in diesem Fall den Atomtyp(mit den Auswahlmöglichkeiten Kohlenstoff(C), Stickstoff(N), Sauerstoff(O), und Schwefel(S).) In der Node-Feature-Matrix sind des weiteren noch die Informationen enthalten, welche formale Ladung das betrachtete Atom hat, und mit wievielen Wasserstoffatomen(H) das Atom(der Knoten) verbunden ist. Außerdem wird noch eine Edge-Feature-Matrix genutzt, um den Typ der chemischen Bindung, zu unterscheiden. In diesem Beispiel wird zwischen Einfach-, Zweifach- und Dreifachbindung unterschieden.

Source: David u. a. (2020)



Algorithm 1 Fragmentation

Require: Molecule M, Fragment List $F \leftarrow []$

```
procedure FRAGMENT(M, F)
    declare Bond b
    b ← GETFIRSTBRICSBOND(M)
    if ISEMPTY(b) then
        return F
    else
        declare Fragment f
        declare Molecule M'
        f, M' ← BREAKMOLATBOND(M, b)
        F ← APPEND(F, f)
        M ← M'
        FRAGMENT(M, F)
    end if
end procedure
```

Abbildung 2.5: In dieser Graphik wird die Zerlegung eines Moleküls in Fragmente mithilfe des BRICS-Algorithmus dargestellt. Podda u. a. (2020)

Source: Podda u. a. (2020)

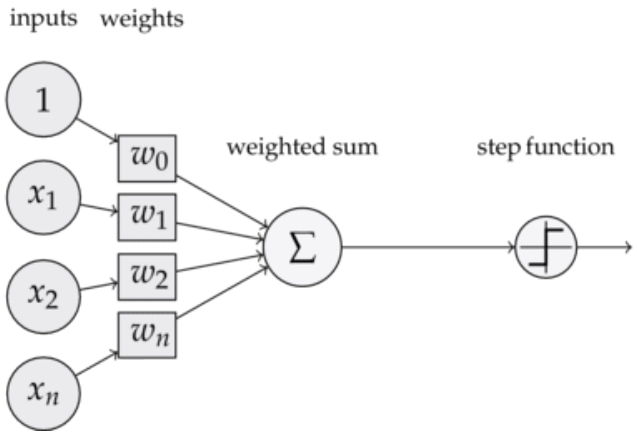


Abbildung 2.6: Hier nochmal eine schematische Darstellung des in Abschnitt 2.3.1 beschriebenen Perceptrons. Die in dieser Abbildung beschriebene Step-Funktion ist genau die, welche durch Formel 2.2 beschrieben wird.

Source: https://www.statworx.com/at/blog/das-rosenblatt-perzeptron-die-fruehen-anfaenge-des-deep-learnings/

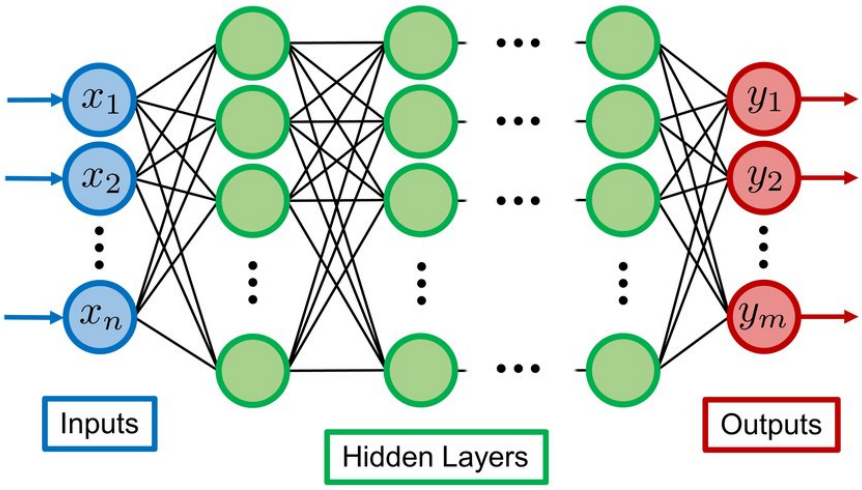


Abbildung 2.7: Hier nochmal eine schematische Darstellung der Feed-Forward neuronalen Netzwerkarchitektur, welche im Abschnitt 2.3.3 beschrieben wird. Das Input Layer, die Hidden-Layer, und das Output Layer sind besonders verdeutlicht.

Source: Kelly u. a. (2021)

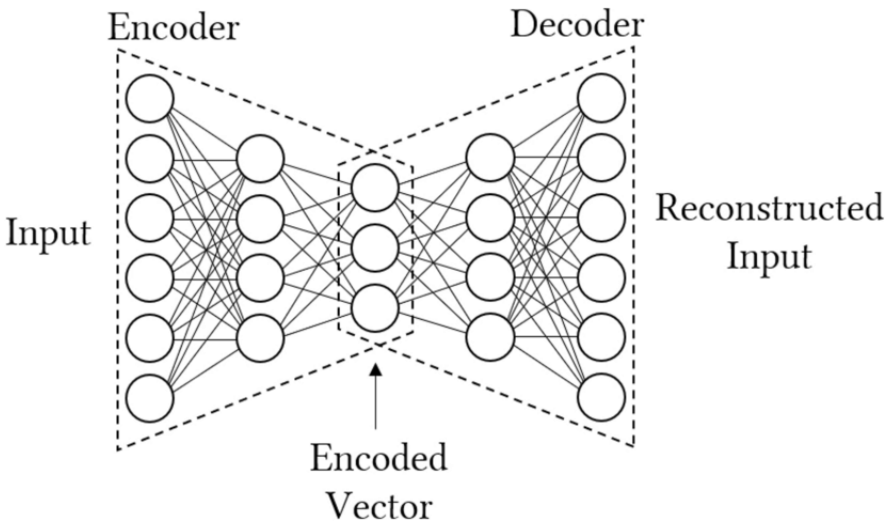


Abbildung 2.8: Eine graphische Darstellung eines *undercomplete Autoencoders*. Der Encoder übersetzt hochdimensionalen Input in niedrigdimensionalen Input, und der Decoder kehrt diesen Prozess um.Watt und Du Plessis (2020)

Source: Watt und Du Plessis (2020)

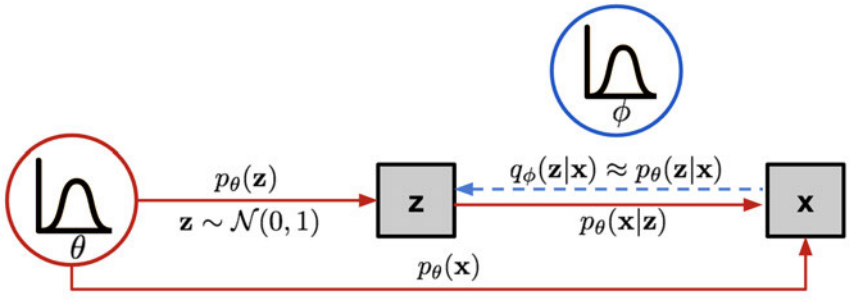


Abbildung 2.9: Eine Darstellung, des graphischen Modells und der strukturellen Beziehungen, der bedingten Wahrscheinlichkeiten/Wahrscheinlichkeitsverteilungen, die hinter variationalen Autoencodern stecken. Weng (2018)

Source: https://lilianweng.github.io/lil-log/assets/images/VAE-graphical-model.png

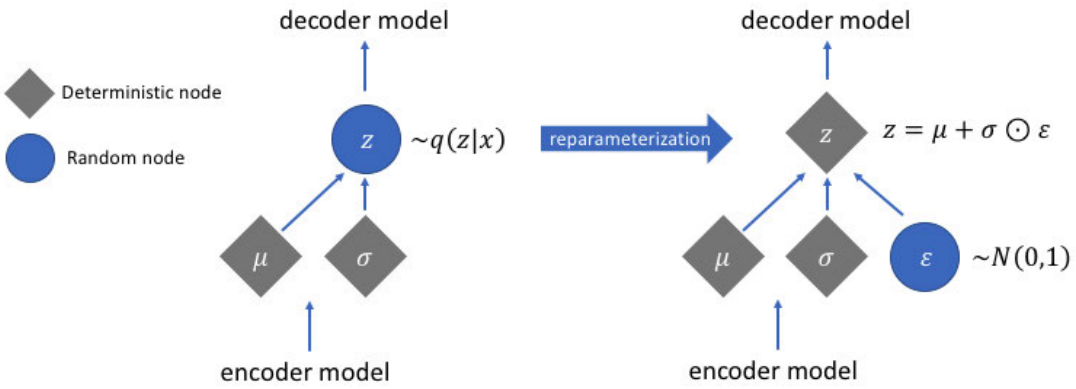


Abbildung 2.10: Eine schematische Darstellung dessen, was beim Reparametrization Trick passiert, um Backpropagation möglich zu machen. Der probabilistische Knoten $\mathbf{z}$ wird deterministisch, indem die Zufälligkeit in die Hilfsvariable ϵ ausgelagert wird.

Source: Jordan (2018)

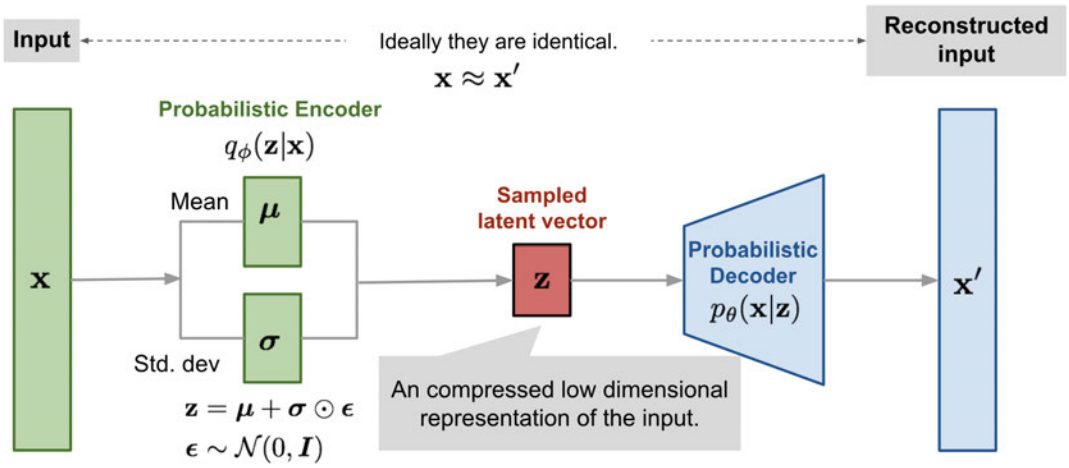


Abbildung 2.11: Eine konzeptionelle Darstellung eines variationalen Autoencoders.

Source: https://lilianweng.github.io/lil-log/assets/images/vae-gaussian.png

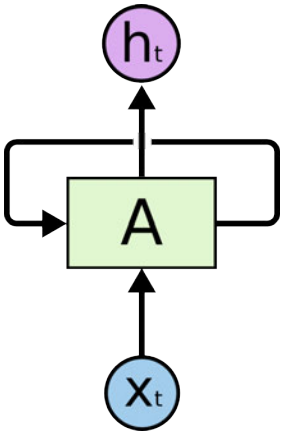


Abbildung 2.12: Eine generelle graphische Darstellung eines rekurrenten neuronalen Netzwerks, mit h_t, als Hidden State und gleichzeitig als Output, und x_t als Input(jeweils schrittabhängig). Hinter A versteckt sich die interne Struktur, des rekurrenten neuronalen Netzes. Die dargestellte Schleife soll zeigen, dass Informationen, von einem Schritt, in den nächsten fließen.Olah (2015)

Source: Olah (2015)

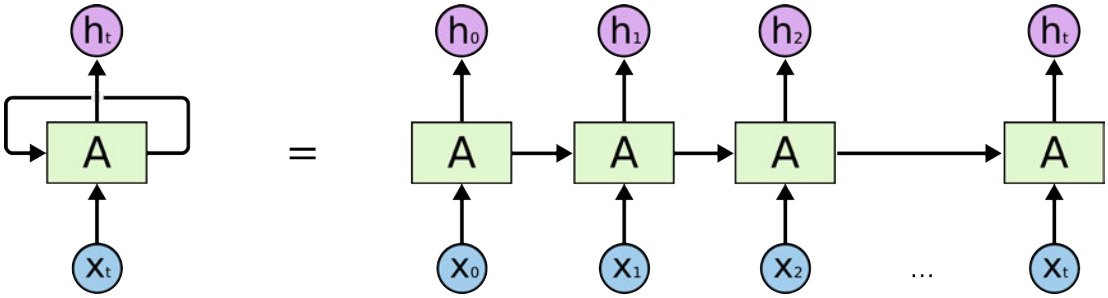


Abbildung 2.13: Die Darstellung aus Abbildung 2.12, umgewandelt, in die aufgeklappte Form. Dadurch wird graphisch deutlich, dass die selbe neuronale Netzwerkstruktur, problemlos variabel lange Sequenzdaten, als Input bekommen kann.Olah (2015)

Source: Olah (2015)

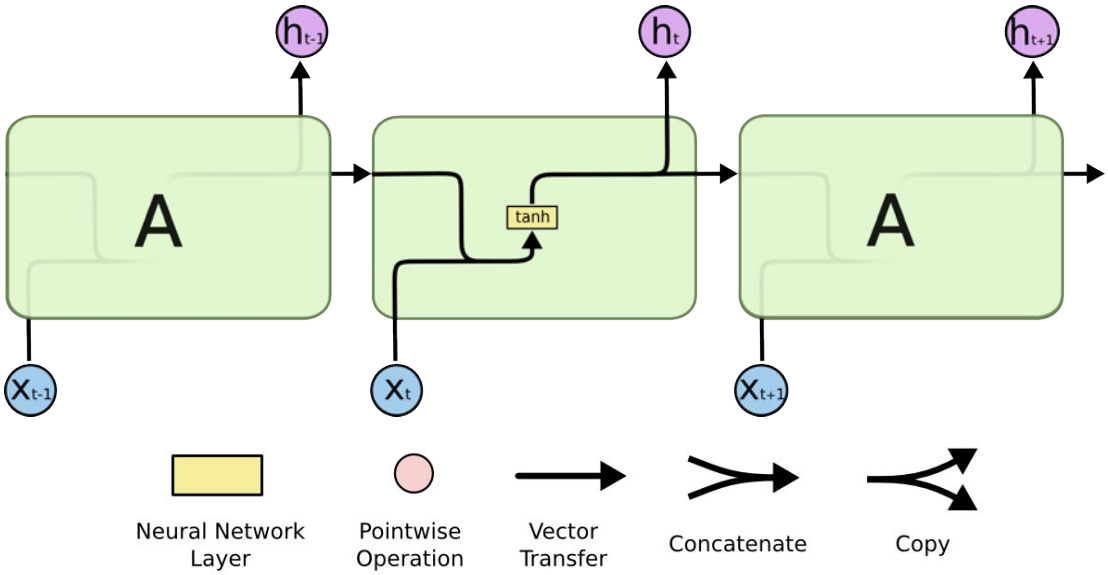


Abbildung 2.14: Eine detailliertere Darstellung von dem, wie die in Abbildung 2.12 und 2.13, vereinfacht dargestellt, intern aufgebaut ist. Es wird graphisch deutlich, dass h_{t-1} und x^t konkateniert werden, und als Input für ein tanh-Layer dienen. Die Ausgabe dieses Layers stellt den Hidden State dar, der an die nächste Generation weitergegeben wird, und gleichzeitig den Output(h_t).Olah (2015)

Source: Olah (2015)

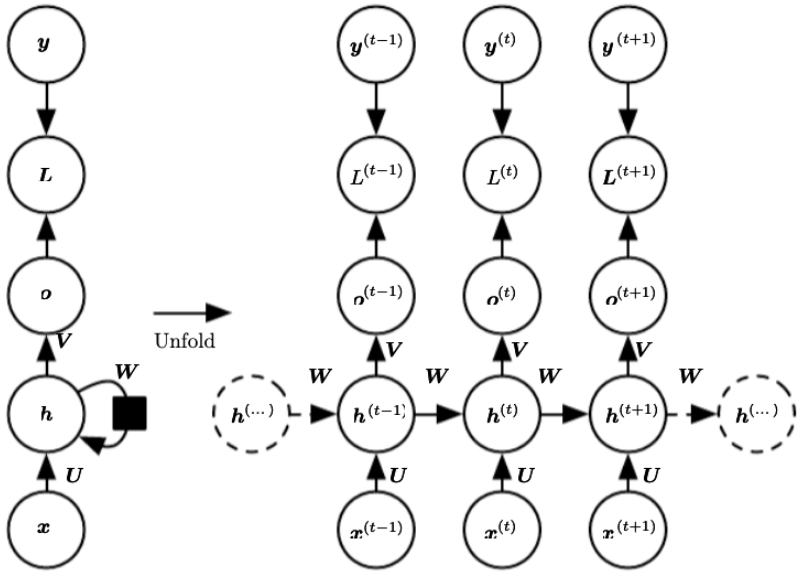


Abbildung 2.15: Eine andere graphische schematische Darstellungsweise von RNNs. x(t) steht für den Input pro Schritt; h(t) steht für den Hidden-State pro Schritt; o(t) steht für den Output pro Schritt; L(t) steht für den loss pro Schritt; y(t) steht für das Label/den erwarteten Output pro Schritt; U, V und W stehen für Gewichtungen/Gewichtsmatrizen (Goodfellow u. a., 2016, Seite 373)

Source: (Goodfellow u. a., 2016, Seite 373)

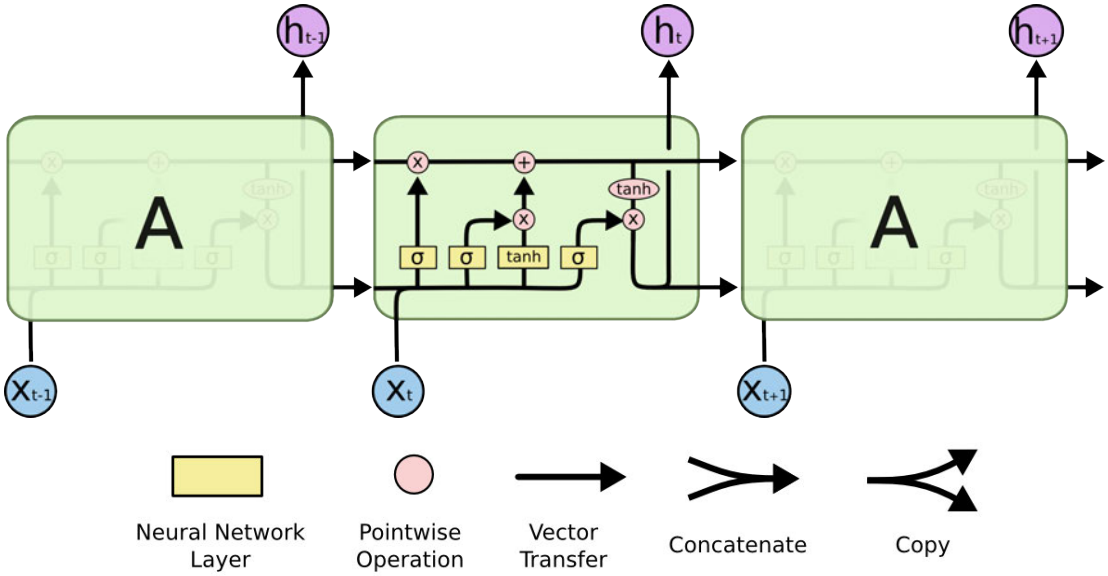


Abbildung 2.16: Die interne Darstellung der Architektur eines LSTMs, mit Legende. Deutlich wird an dieser Stelle schon, aus welchen grundlegenden Layern, das LSTM besteht, und welche mathematischen Operationen in diesem vorgenommen werden. Des weiteren wird auch schon deutlich, dass hier Informationen an zwei Stellen von einer LSTM-Zelle, an die nächste übergeben werden.Olah (2015)

Source: Olah (2015)

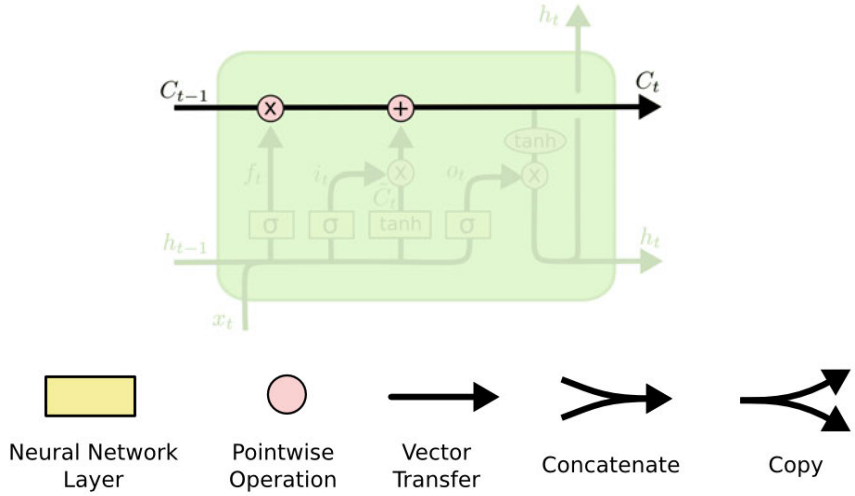


Abbildung 2.17: Die interne Darstellung der Architektur eines LSTMs, mit Fokus auf die Weiterleitung des Cell States. Dabei wird deutlich, dass nur eine punktweise Multiplikation, und eine punktweise Addition vorgenommen wird, um den Cell State für den jeweils folgenden Schritt zu bekommen. Olah (2015)

Source: Olah (2015)

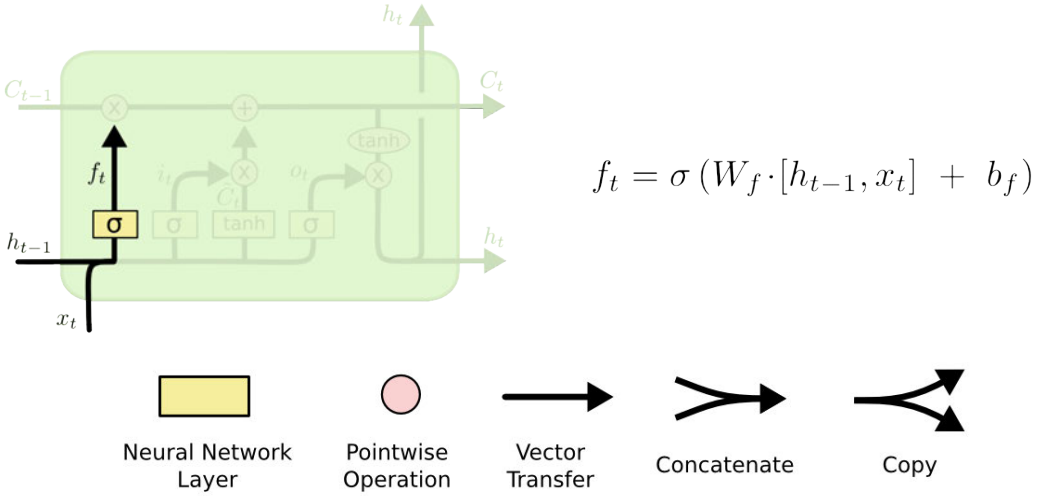


Abbildung 2.18: Die interne Darstellung der Architektur eines LSTMs, mit Legende, mit Fokus auf das Forget Gate. Dieses wird als f_t bezeichnet, und die Berechnung von f_t wird ebenfalls dargestellt. Es handelt sich dabei um ein Sigmoid-Layer(dargestellt durch das σ). In diesem Zusammenhang steht W_f für den Gewichtsvektor; und b_f für die Biasvektor.Olah (2015)

Source: Olah (2015)

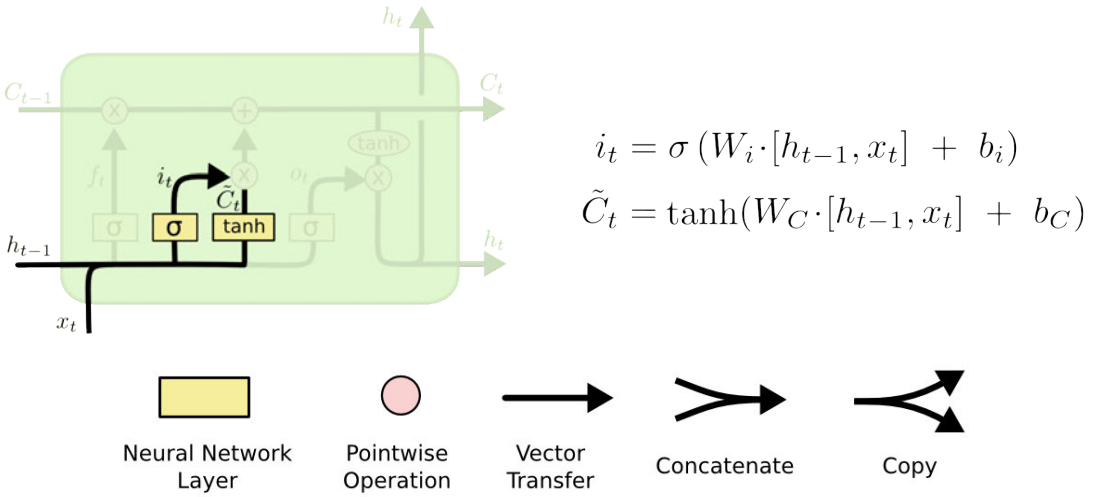


Abbildung 2.19: Die interne Darstellung der Architektur eines LSTMs, mit Legende, mit Fokus auf einen Teil des Input-Gates, mit Formeln für i_t und $\tilde{C}_t$. Das dargestellte i_t stellt schematisch die Gewichtung der neu gelernten Informationen dar, umgesetzt durch ein Sigmoid-Layer. Das $\tilde{C}_t$ steht inhaltlich für die neuen Informationen, die aus der Konkatenation des Hidden States, des vorigen Schrittes, und des Inputs des aktuellen Schrittes, entnommen werden sollen, und im nachfolgenden, dem Cell State hinzugefügt werden sollen.Olah (2015)

Source: Olah (2015)

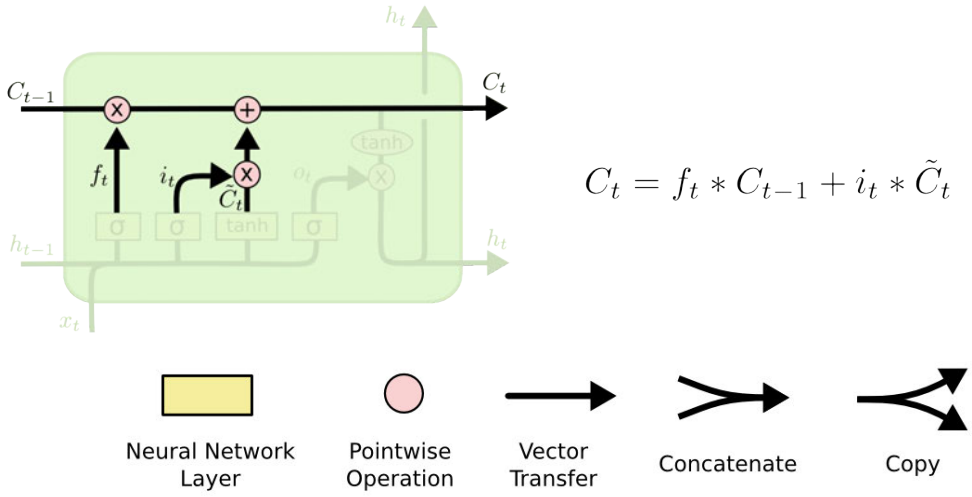


Abbildung 2.20: Die interne Darstellung der Architektur eines LSTMs, mit Legende, mit Fokus darauf wie der Cell State genau, von einem Schritt zum nächsten geupdatet werden soll, mit Formel für $\tilde{C}_t$, wodurch die Informationen aus den Abbildungen 2.17, 2.18, 2.19, und 2.20, zusammenkommen. Olah (2015)

Source: Olah (2015)

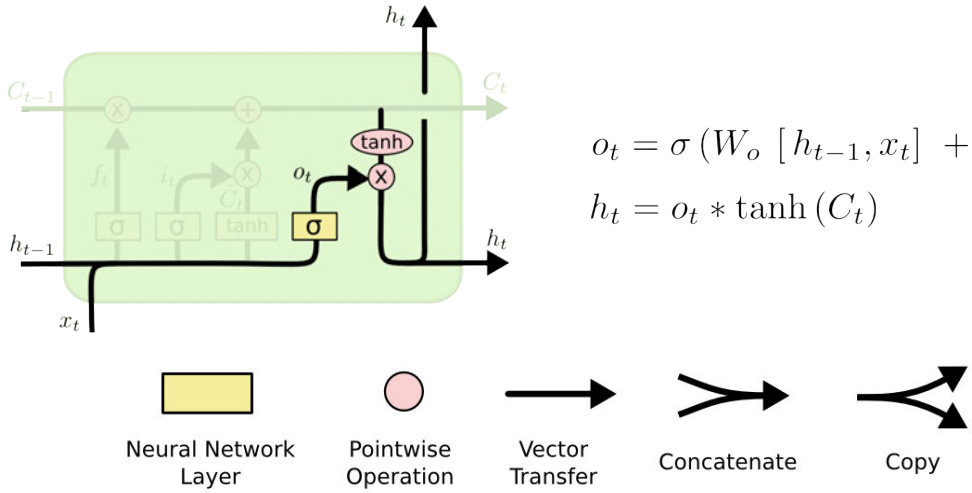


Abbildung 2.21: Die interne Darstellung der Architektur eines LSTMs, mit Legende, mit Fokus auf das o_t-Layer, und der Berechnung des Hidden States(h_t), mit graphischer Darstellung, und zugrundeliegenden Formeln.Olah (2015)

Source: Olah (2015)

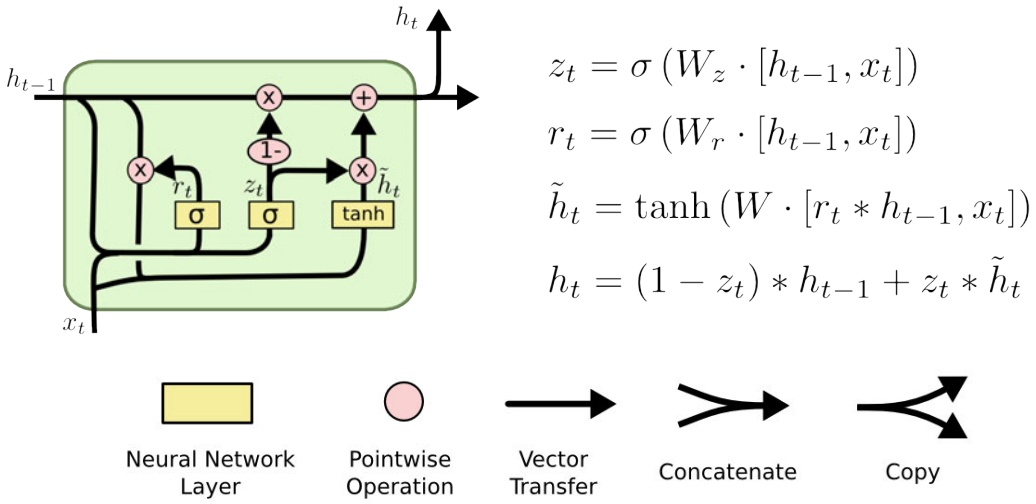


Abbildung 2.22: Die interne Darstellung der Architektur eines GRUs, mit Legende, mit graphischer Darstellung, und zugrundeliegenden Formeln. Bestehend aus zwei Sigmoid Layern(z_t und r_t), und einem Layer mit dem Tangens Hyperbolicus, als Aktivierungsfunktion(das Layer ist als $\tilde{h}_t$ notiert). Dieses nimmt als Input die Konkatention des mit r_t gewichteten Hidden States, des Schrittes zuvor(h_{t-1}), konkatentiert mit dem Input des aktuellen Schrittes(x_t), als Input. Des weiteren wird mit der Formel für h_t ersichtlich, wie sich der Hidden State(h_t), für jeden Schritt berechnen lässt. Olah (2015)

Source: Olah (2015)

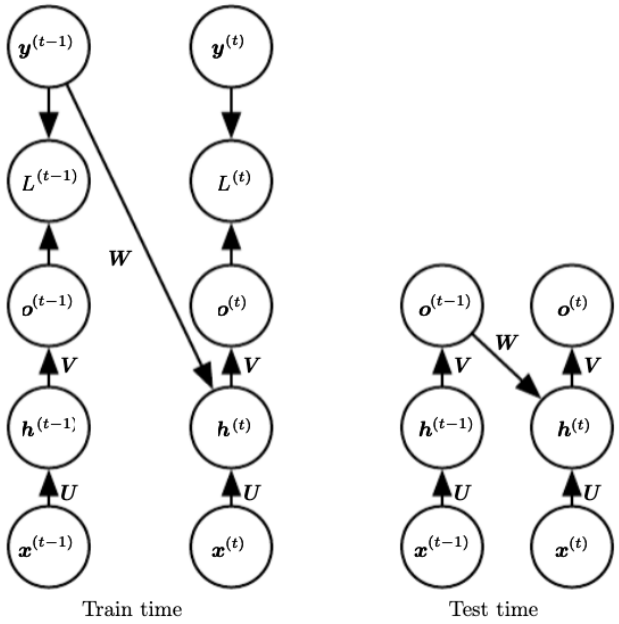


Abbildung 2.23: Eine Darstellung der Vorgehensweise des Teacher Forcings. Dabei wird ersichtlich, dass während des Trainings, der erwünschte Output(y^{t-1}), des vorherigen Schrittes genommen wird, anstatt des tatsächlich generierten Outputs, des vorigen Schrittes(o^{t-1}). Nach dem Training(in der Graphik als *Test time* betitelt), wird aber der tatsächlich generierten Outputs, des vorigen Schrittes(o^{t-1}) weitergegeben. x(t) steht für den Input pro Schritt; h(t) steht für den Hidden-State pro Schritt; o(t) steht für den Output pro Schritt; L(t) steht für den loss pro Schritt; y(t) steht für das Label/den erwarteten Output pro Schritt; U, V und W stehen für Gewichtungen/Gewichtsmatrizen(Goodfellow u. a., 2016, Seite 377)

Source: (Goodfellow u. a., 2016, Kapitel 10, 10.2.1, Seite 377)

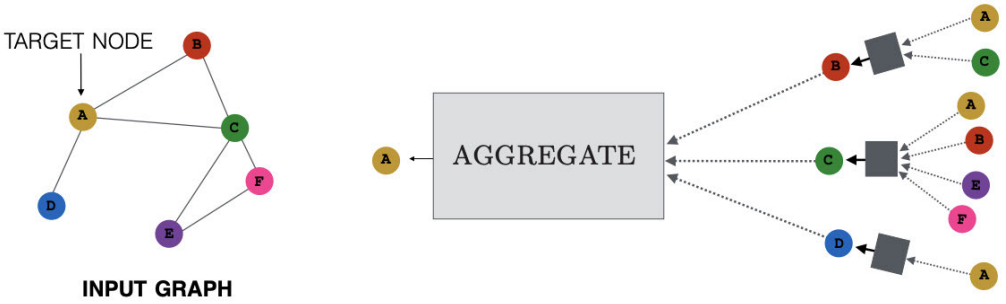


Abbildung 2.24: Eine schematische graphische Darstellung dessen, wie ein Knoten die Nachrichten(messages), in der AGGREGATE-Funktion zusammenführt. Deutlich wird auch, dass die Berechnungstruktur für den Zustand eines Knotens, eine Baumstruktur annimmt. Hamilton (2020)

Source: Hamilton (2020)

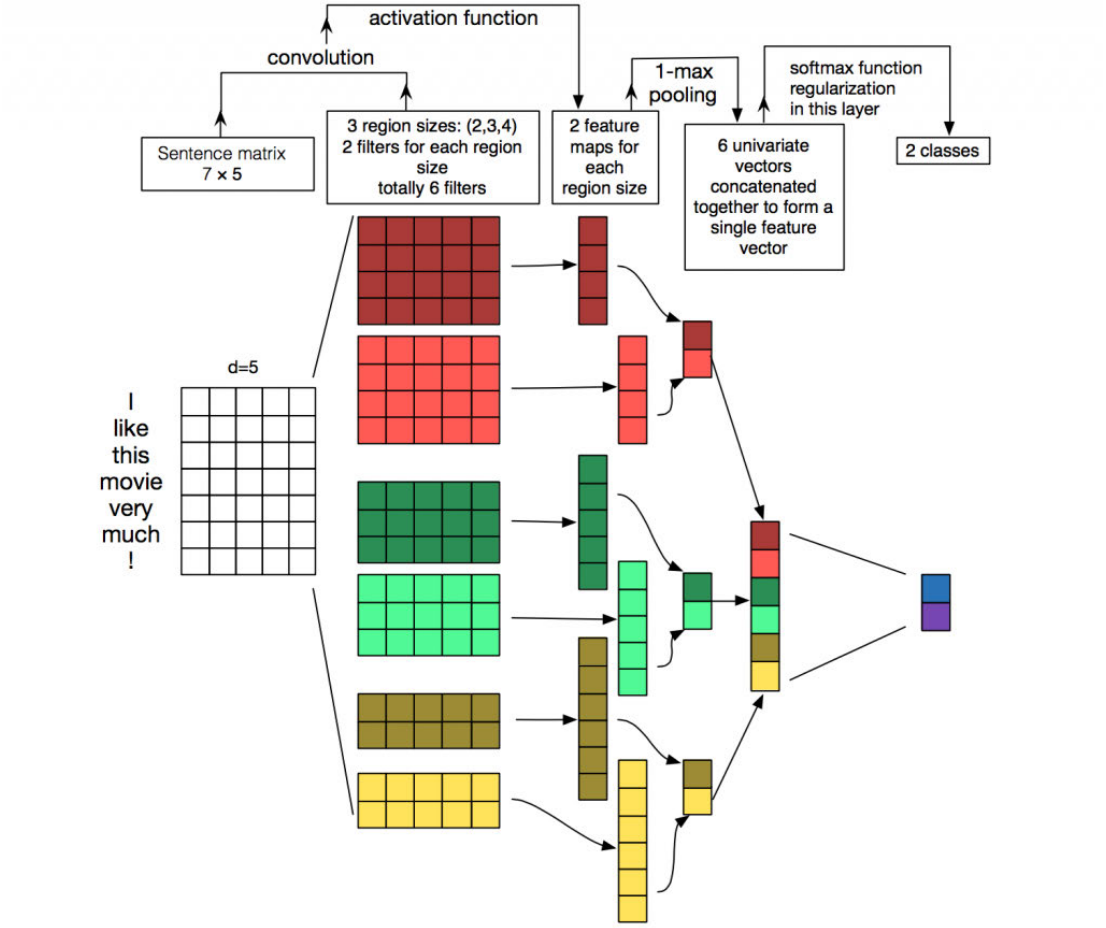


Abbildung 2.25: Ein Beispiel für ein CNN für eine Satz-Klassifikations-Problem. Die genutzen Filter werden in drei verschiedenen Regions-Größen(2,3,4) genutzt, welche über alle Embedding-Dimensionen gehen, und von der selben Größe werde immer jeweils zwei verschiedene Filter genutzt. Das führt dazu, dass jeweils zwei Feature-Maps pro Filter-Größe produziert werden. Es wird eine schmale Convolution angewandt. Die Filter wenden Gewichtungen + eine Aktivierungsfunktion auf ihre Werte an. Danach wird Max-Pooling angewandt, und von jedem Filter wird nur der Maximalwert weiter verwendet. Nach einem Softmax-Layer erhält man einen zweidimensionalen Output, welcher dazu da ist den Input in zwei Klassen einzuordnen, oder nicht. Britz (2015); Zhang und Wallace (2016)

Source: Zhang und Wallace (2016)

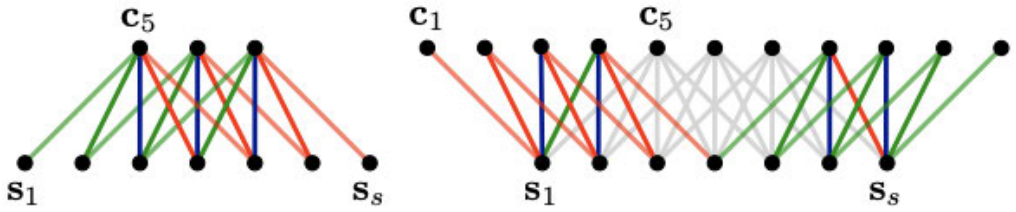


Abbildung 2.26: Eine graphische Darstellung schmaler Convolutionen(linke Teilabbildung) im Vergleich zu einer breiten Convolution(rechte Teilabbildung). In beiden Fällen ist die Filter-Größe $m = 5$. Kalchbrenner u. a. (2014)

Source: Kalchbrenner u. a. (2014)

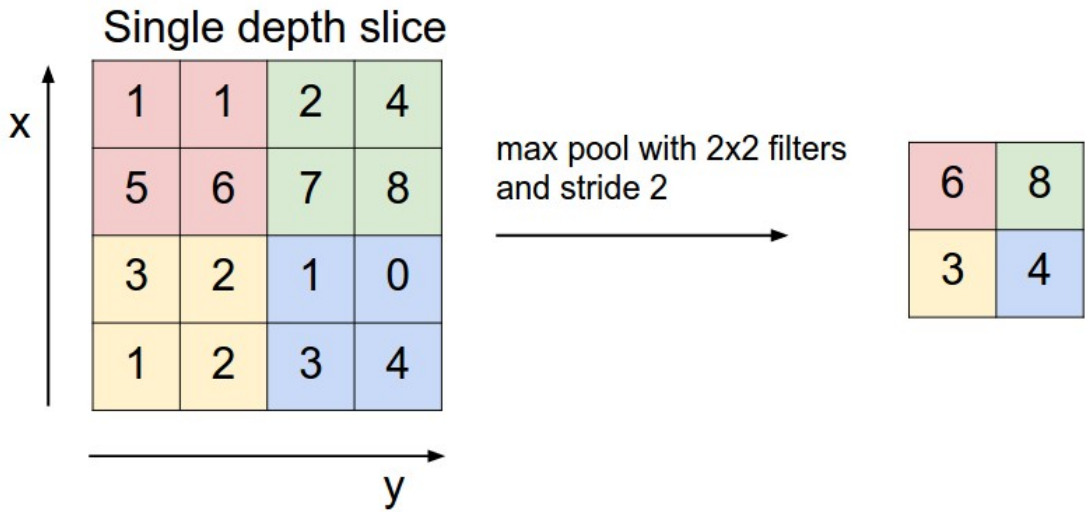


Abbildung 2.27: Ein graphisches Beispiel für Max-Pooling. Hier dargestellt für ein 2D-CNN. Der Stride ist in diesem Fall 2, und die Filter sind 2×2 groß. Konzeptionell besteht hier jedoch kein Unterschied zwischen diesem zweidimensionalen Beispiel, nur dass Input + Filter + Output auch eindimensional wären. Britz (2015)

Source: https://cs231n.github.io/convolutional-networks/

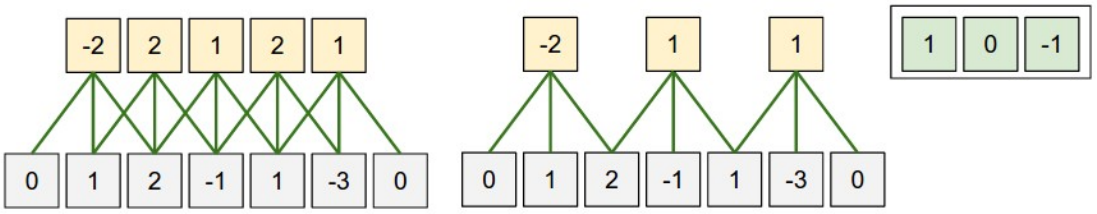


Abbildung 2.28: Ein graphisches Beispiel für ein 1D-CNN, einmal mit einer Stride Size von 1(linkes Beispiel), und einmal mit einer Stride Size von 2(mittleres Beispiel). Ganz rechts ist der angewandte Filter dargestellt. Britz (2015)

Source: https://cs231n.github.io/convolutional-networks/

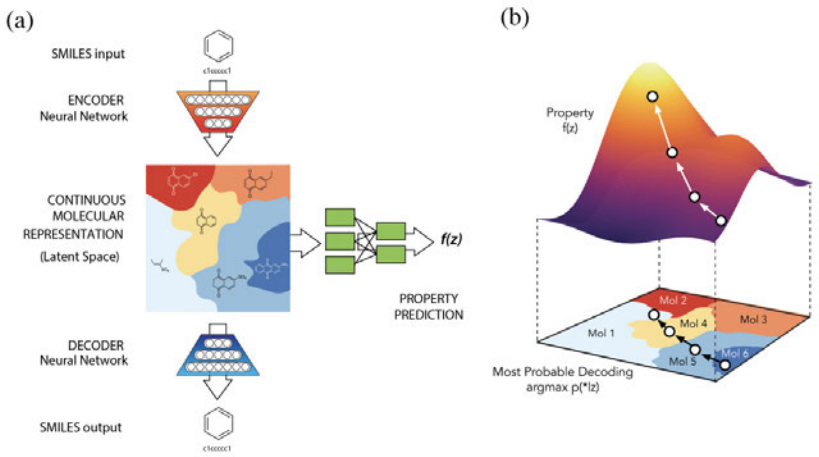


Abbildung 2.29: Ein Überblick über die beschriebene variationale Autoencoder-Architektur. Es wird sowohl in (a) und (b) angedeutet, dass der latent Space sich nach bestimmten Eigenschaften ordnet. Gómez-Bombarelli u. a. (2018)

Source: Gómez-Bombarelli u. a. (2018)

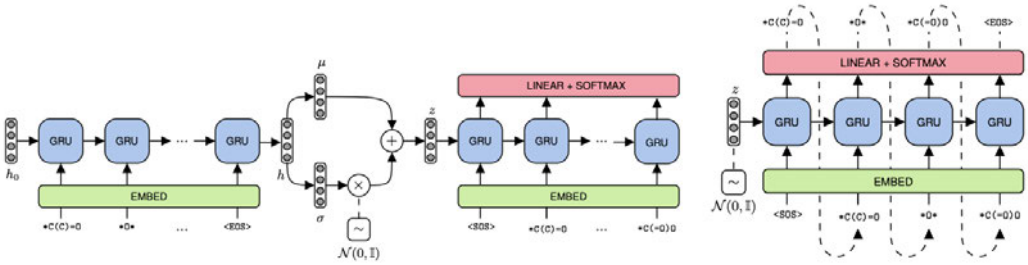


Abbildung 2.30: Eine zusammengefasste Architektur des Fragment-based VAE. Die einzelnen beschriebenen Komponenten werden deutlich. Das Embedding-Layer; der aus GRUs bestehende Encoder; der Einsatz des Reparametrization Tricks; der Decoder(bestehend aus GRUs mit einem Softmax-Layer). Des weiteren wird durch (a) und (b) deutlich, das hier wie beschrieben, Teacher Forcing genutzt wird.

Source: Podda u. a. (2020)

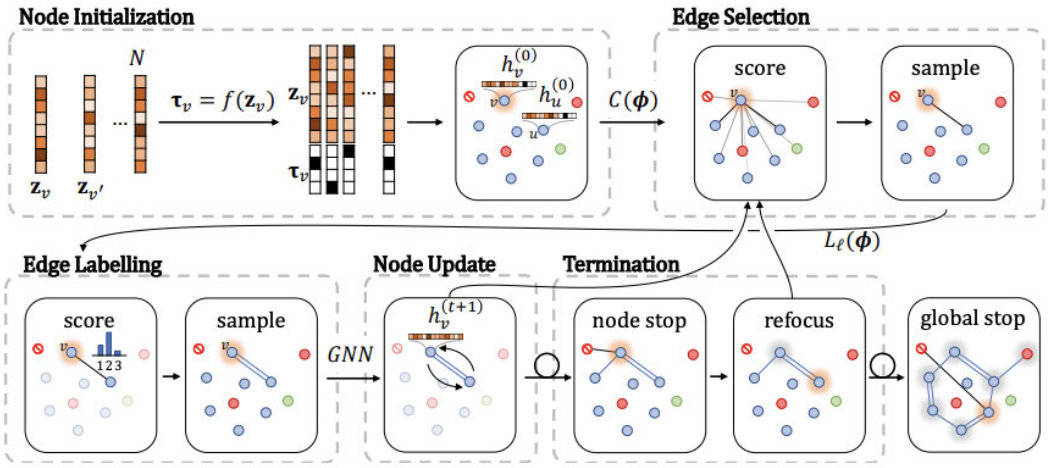


Abbildung 2.31: Der im Folgenden Abschnitt beschriebene Generierungsprozess, graphisch dargestellt. Die Phasen Knoten-Initialisierung(Node Initialization), Kanten-Auswahl(Edge Selection), Kanten-Labelling(Edge Labelling), Knoten-Update(Node Update), und die Termination der einzelnen Phasen, und das Ende des gesamten Generierungsprozesses sind hier detailliert dargestellt. Ein Pfeil mit Schleife soll darstellen, dass Teilprozesse mehrfach ablaufen. Liu u. a. (2018)

Source: Liu u. a. (2018)

3 Methoden

Die betrachteten Paper/VAE-Konzepte:

- ChemVAE:
  - Paper: Gómez-Bombarelli u. a. (2018)
  - Github-Repository: https://github.com/aspuru-guzik-group/chemical_vae
- Fragment-based-VAE:
  - Paper: Podda u. a. (2020)
  - Github-Repository: https://github.com/marcopodda/fragment-based-dgm
- CGVAE:
  - Paper: Liu u. a. (2018)
  - Github-Repository: https://github.com/microsoft/constrained-graph-var

Der verwendete Datensatz zum trainieren der VAEs war der Zinc-Datensatz, welcher aus etwas weniger als 250.000 verschiedenen SMILES-Strings besteht. Dieser ist ein oft verwendeter Datensatz zum Benchmarking von generativen Modellen für Moleküle.

Aus dem Latent Space jeder der betrachteten VAE-Architekturen wurden 10.000 Punkte gesampelt, und diese wurden dann von dem jeweiligen Decoder decodet. Das Ergebnis besteht aus 10.000 Strings im SMILES-Format. Dann wird für jede VAE-Architektur ermittelt, wie viele der SMILES-Strings valide, neuartige und einzigartige Moleküle darstellen.

- **Valide Moleküle**: Ein Molekül wird dann als valide(engl. valid) angesehen, wenn der SMILES-String, welcher es darstellen soll, von einer Library wie Rdkit(http://www.rdkit.org)Landrum u. a. (2020) für valide befunden wird. Semantisch steckt dahinter, dass das in dem SMILES-String beschriebene organische Molekül auch real existieren kann. Der andere Fall wäre, dass der SMILES-String zwar aus Zeichen besteht, die erlaubt

sind im SMILES-Format, diese aber in einer Form kombiniert sind, die sich nicht in eine real mögliche Molekülstruktur überführen lassen. Podda u. a. (2020)

- **Neuartige Moleküle**: Ein Molekül wird dann als neuartig(engl. novel) betrachtet, wenn es nicht im Originaldatensatz/Trainingsdatensatz enthalten ist. Diese Überprüfung dient der Untersuchung„ ob das Model nicht nur den Reconstruction Error minimiert ohne den Latent Space effektiv auszufüllen. Podda u. a. (2020) Dies ist eine wichtige Metrik, um festzustellen, ob der variationale Autoencoder sich gut als generatives Model für Moleküle eignet.

- **Einzigartige Moleküle**: Ein Molekül gilt dann als einzigartig(engl. unique) wenn es im Zuge des Sampling-Prozesses, nur einmal als Output decodet wird. Podda u. a. (2020) Dies soll sicherstellen, dass der jeweilige variationale Autoencoder Teile des Latent Spaces mit wenigen Molekülen ausfüllt. Optimalerweise sind ein Großteil der gesampelten Moleküle einzigartig. Das wäre ein Zeichen dafür, dass der betrachtete VAE sich gut als generatives Model eignet.

Des Weiteren werden strukturabgeleitete Eigenschaften betrachtet. Das sind Eigenschaften, die sich für ein Molekül gegeben seines SMILES-Strings mithilfe von Libraries wie Rdkit(`http://www.rdkit.org`)Landrum u. a. (2020) berechnen lassen. Die beschriebenen Evaluationen wurden zum Teil mit dem *evaluate.py*-Skript aus dem CGVAE-Github-Repository ermittelt. Im Fall des Fragment-based VAE wurde die Evaluation mithilfe des sample-Befehls ermittelt.

Die betrachteten Eigenschaften sind:

- Quantitative Estimate of Drug-likeness(QED)
- der Synthetic Accessibility Score(SAS)
- und der LogP-Wert

Die 10.000 gesampelten Moleküle wurden je nach Architektur auf verschiedenen Wegen ermittelt. Für ChemVAE waren im bereitgestellten Github-Repository bereits vortrainierte neuronale Netzwerke vorhanden. Das Model, welches im Ordner *models/zinc* enthalten war, wurde verwendet. (Dieses wurde allerdings ohne den erwähnten Property-Predictor trainiert.) Der VAE wurde für 70 Epochs auf dem Zinc-Datensatz trainiert. Die Config-File des pretrained Models wird zur Verfügung gestellt. Der Fragment-based-VAE wurde eigens trainiert. Die Config-File des Trainings wird zur Verfügung gestellt. Der CGVAE konnte aufgrund eines, unter gegebenen Umständen nicht erbringbaren operationellen Aufwandes, nicht selber trainiert werden.

Die 10.000 generierten Moleküle wurden freundlicherweise von den Autoren des Papers Liu u. a. (2018) zur Verfügung gestellt.

Es wurden für jede dieser Eigenschaften wurde die Wahrscheinlichkeitsdichteverteilung des Zinc-Datensatzes, über die Wahrscheinlichkeitsdichteverteilung der Eigenschaften, aus den jeweils 10.000 generierten Molekülen gelegt. Diese Wahrscheinlichkeitsdichteverteilungen wurden als Liniendiagramme dargestellt. Dies wurde mithilfe der in `https://github.com/marcopodda/fragment-based-dgm` enthaltenen Postprocessing- und Plot-Befehle erreicht. Einige kleine lokale Änderungen des Codes waren nötig, um die generierten SMILES, von allen Projekten mithilfe dieser Funktionalität so analysieren zu können.

Des weiteren wurden die durchschnittlich Anzahl bestimmter struktureller Eigenschaften bestimmt. Diese wurden dann als gestapelte Balkendiagramme(engl. stacked bar charts) dargestellt. Dies wurde einmal für den ZINC-Datensatz getan, und je einmal für die jeweils 10.000 generierten SMILES-Strings. Die strukturellen Eigenschaften sind in drei Gruppen unterteilt: Atome, Bindungen und Ringe.

Die durchschnittliche Anzahl der pro Molekül enthaltenen Atome(pro Element) wurde für die folgendene Elemente gezählt:

- Kohlenstoff(C)
- Fluor(F)
- Stickstoff(N)
- Sauerstoff(O)
- Des Weiteren werden alle anderen Atome in einer weiteren Kategorie(andere(engl. other)) zusammengefasst.

Die Bindungen beziehungsweise Bindungstypen, die gezählt werden sind Einfach-, Zweifach- und Dreifachbindungen. Die Ringe beziehungsweise Ringtypen die gezählt werden sind Dreierringe, Viererringe, Fünferringe und Sechserringe. Die vorgestellten durchschnittlichen Anzahlen der Struktureigenschaften werden durch Balkendiagramme der erwähnter Gruppe dargestellt. Diese wurden ebenfalls mithilfe der in `https://github.com/marcopodda/fragment-based-dgm` enthaltenen Postprocessing- und Plot-Befehle generiert.

Die beschrieben, graphischen Darstellungen wurden generiert, um darzustellen, wie gut die betrachteten variationalen Autoencoder die Eigenschaftenverteilungen der Originaldaten lernen.

4 Ergebnisse

	% valide Moleküle	% neuratige Moleküle	% einzigartige Moleküle
ChemVAE	0,24	100,00	100,00
Fragment-based-VAE	99,99	97,06	12,78
CGVAE	100,00	99,97	100,00

Tabelle 4.1: Die Ergebnisse der Auswertung: Der prozentuale Anteil der validen, neuartigen und einzigartigen Moleküle, von 10.000 generierten SMILES, pro betrachteter VAE-Architektur.

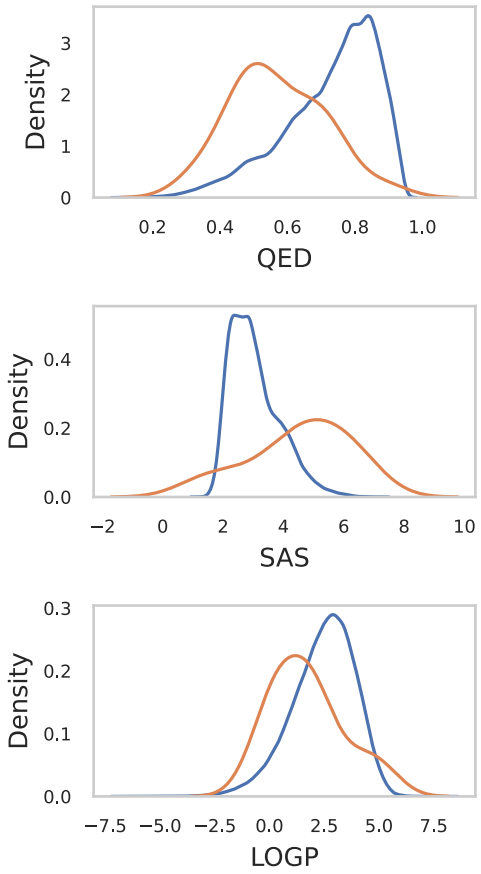


Abbildung 4.1: **ChemVAE**: Die Eigenschaftsverteilungen des Zinc-Datensatzes und den Eigenschaftsverteilungen aus 10.000 zufällig gesampelten Punkten aus dem Latent Space. Dabei ist die Eigenschaftsverteilung des Zinc-Datensatzes in blau dargestellt und die, die vom ChemVAE generiert wurde, in gelb. Die betrachteten Eigenschaften sind der Quantitative Estimate of Drug-likeness(QED), der Synthetic Accessibility Score(SAS) und der LogP-Wert. Die Eigenschaften sind in den Subplots jeweils auf der x-Achse aufgetragen. Die y-Achse stellt in den Subplots die Wahrscheinlichkeitsdichte dar.

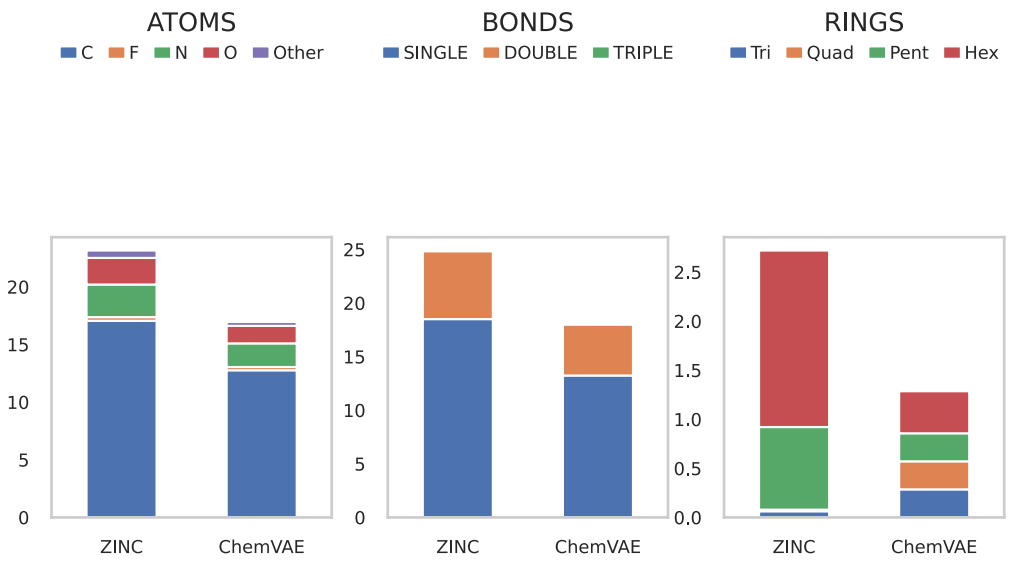


Abbildung 4.2: **ChemVAE**: Die jeweiligen durchschnittlichen Anzahlen pro Molekül, der verschiedener struktureller Merkmale, des Zinc-Datensatzes(als ZINC notiert), im Vergleich zu den durchschnittlichen Anzahlen der strukturellen Merkmale der Moleküle, die durch ChemVAE generiert wurden, graphisch dargestellt. Dieser Plot ist in drei Subplots aufgeteilt. Der erste befasst sich mit den durschnittlichen Anzahlen einiger wichtiger Atome. Dabei werden Kohlenstoff(C), Fluor(F), Stickstoff(N) und Sauerstoff(O) gesondert aufgeführt, und alle anderen unter *Other* zusammengefasst. Im zweiten Subplot wird das durchschnittliche Vorkommen von Bindungen pro Molekül dargestellt. Die verschiedenen betrachteten Bindungstypen sind Einfach-, Zweifach-, und Dreifachbindungen. Im dritten Subplot werden die durchschnittlichen Anzahlen verschiedener Ringtypen im Molekül dargestellt. Es wird dabei unter Dreierringen(Tri), Viererringen(Quad), Fünferringen(Pent) und Sechserringen(Hex) unterschieden.

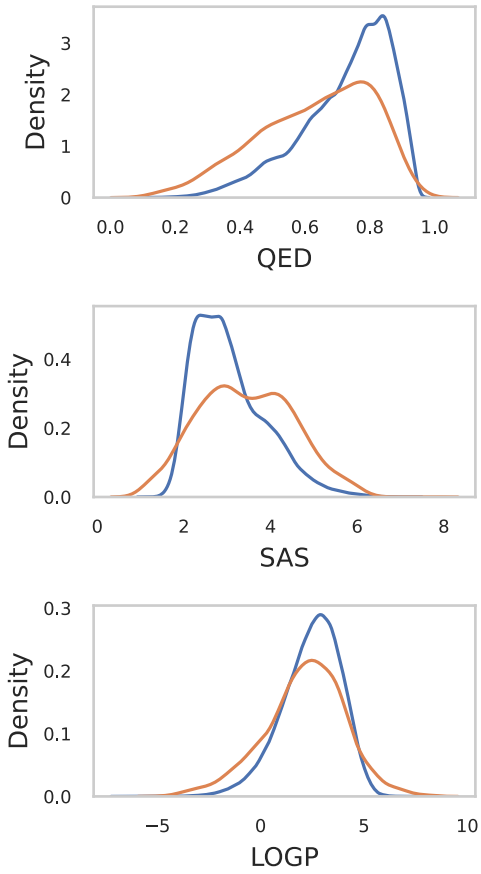


Abbildung 4.3: **Fragment-based-VAE**: Die Eigenschaftsverteilungen des Zinc-Datensatzes und den Eigenschaftsverteilungen aus 10.000 zufällig gesampelten Punkten aus dem Latent Space. Dabei ist die Eigenschaftsverteilung des Zinc-Datensatzes in blau dargestellt und die, die vom Fragment-based-VAE generiert wurde, in gelb. Die betrachteten Eigenschaften sind der Quantitative Estimate of Drug-likeness(QED), der Synthetic Accessibility Score(SAS) und der LogP-Wert. Die Eigenschaften sind in den Subplots jeweils auf der x-Achse aufgetragen. Die y-Achse stellt in den Subplots die Wahrscheinlichkeitsdichte dar.

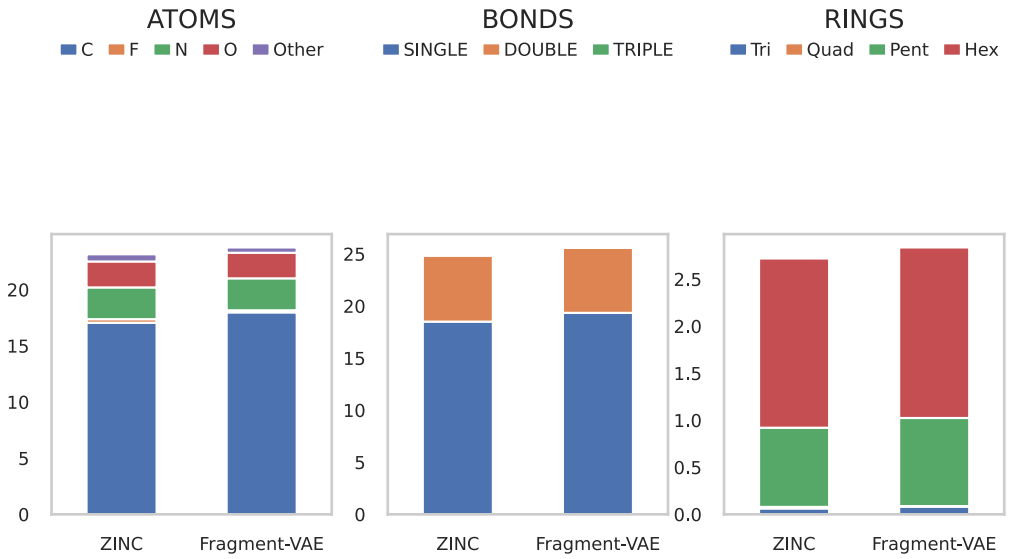


Abbildung 4.4: **Fragment-based-VAE**: Die jeweiligen durchschnittlichen Anzahlen pro Molekül, der verschiedener struktureller Merkmale, des Zinc-Datensatzes(als ZINC notiert), im Vergleich zu den durchschnittlichen Anzahlen der strukturellen Merkmale der Moleküle, die durch Fragment-based-VAE generiert wurden, graphisch dargestellt. Dieser Plot ist in drei Subplots aufgeteilt. Der erste befasst sich mit den durschnittlichen Anzahlen einiger wichtiger Atome. Dabei werden Kohlenstoff(C), Fluor(F), Stickstoff(N) und Sauerstoff(O) gesondert aufgeführt, und alle anderen unter *Other* zusammengefasst. Im zweiten Subplot wird das durchschnittliche Vorkommen von Bindungen pro Molekül dargestellt. Die verschiedenen betrachteten Bindungstypen sind Einfach-, Zweifach-, und Dreifachbindungen. Im dritten Subplot werden die durchschnittlichen Anzahlen verschiedener Ringtypen im Molekül dargestellt. Es wird dabei unter Dreierringen(Tri), Viererringen(Quad), Fünferringen(Pent) und Sechserringen(Hex) unterschieden.

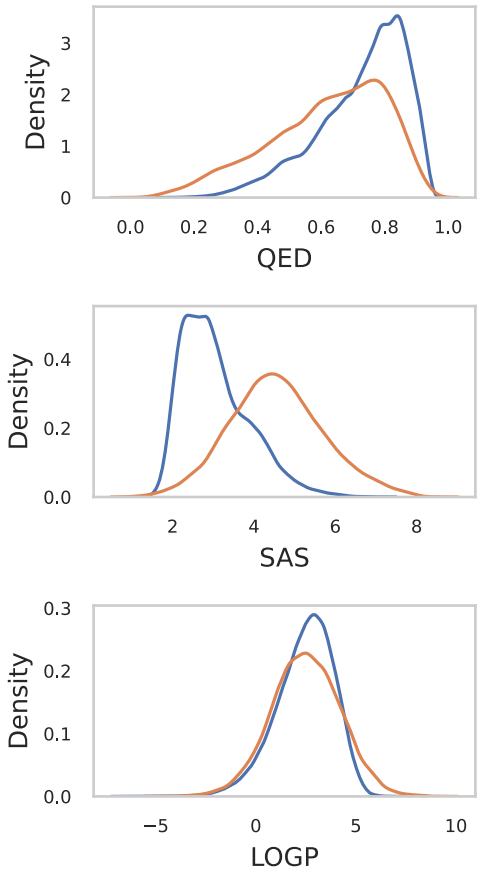


Abbildung 4.5: **CGVAE**: Die Eigenschaftsverteilungen des Zinc-Datensatzes und den Eigenschaftsverteilungen aus 10.000 zufällig gesampelten Punkten aus dem Latent Space. Dabei ist die Eigenschaftsverteilung des Zinc-Datensatzes in blau dargestellt und die, die vom CGVAE generiert wurde, in gelb. Die betrachteten Eigenschaften sind der Quantitative Estimate of Drug-likeness(QED), der Synthetic Accessibility Score(SAS) und der LogP-Wert. Die Eigenschaften sind in den Subplots jeweils auf der x-Achse aufgetragen. Die y-Achse stellt in den Subplots die Wahrscheinlichkeitsdichte dar.

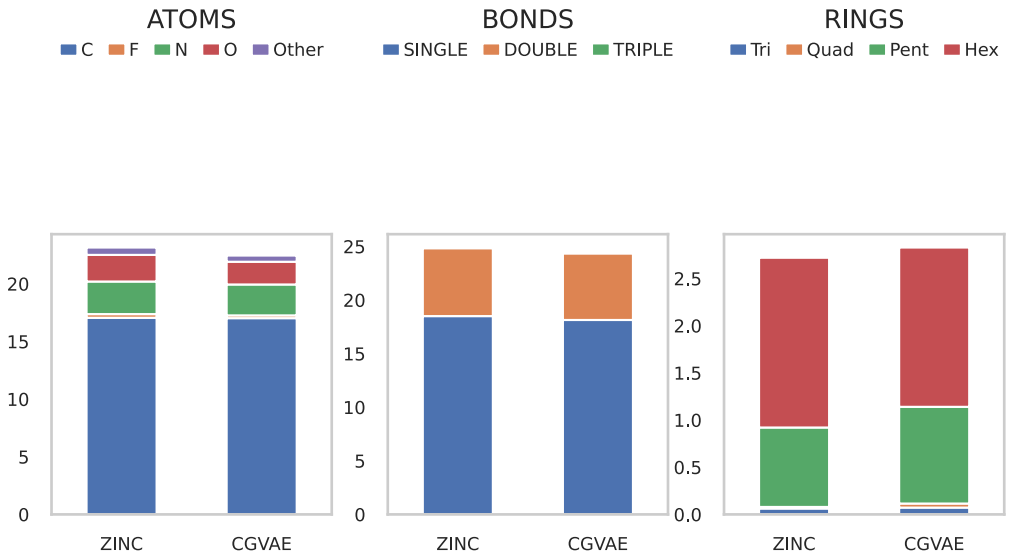


Abbildung 4.6: **CGVAE**: Die jeweiligen durchschnittlichen Anzahlen pro Molekül, der verschiedener struktureller Merkmale, des Zinc-Datensatzes(als ZINC notiert), im Vergleich zu den durchschnittlichen Anzahlen der strukturellen Merkmale der Moleküle, die durch CGVAE generiert wurden, graphisch dargestellt. Dieser Plot ist in drei Subplots aufgeteilt. Der erste befasst sich mit den durschnittlichen Anzahlen einiger wichtiger Atome. Dabei werden Kohlenstoff(C), Fluor(F), Stickstoff(N) und Sauerstoff(O) gesondert aufgeführt, und alle anderen unter *Other* zusammengefasst. Im zweiten Subplot wird das durchschnittliche Vorkommen von Bindungen pro Molekül dargestellt. Die verschiedenen betrachteten Bindungstypen sind Einfach-, Zweifach-, und Dreifachbindungen. Im dritten Subplot werden die durchschnittlichen Anzahlen verschiedener Ringtypen im Molekül dargestellt. Es wird dabei unter Dreierringen(Tri), Viererringen(Quad), Fünferringen(Pent) und Sechserringen(Hex) unterschieden.

5 Auswertung

5.1 Die Tabelle

5.1.1 ChemVAE

Die ermittelte Validität des betrachteten ChemVAE ist mit 0,24% schlecht für zufällig gesampelte Punkte aus dem Latent Space. Vergleicht man diesen Wert mit den bestmöglichen Werten für zufällig gesampelte Werte, dann erreicht Gómez-Bombarelli u. a. (2018) unter besten Bedingungen 4%. Es gibt mehrere mögliche Erklärungen für dieses Ergebnis. Eine mögliche Erklärung ist, dass das pretrained Model, was im öffentlichen Github-Repository bereitgestellt wurde, nicht mit optimalen Hyperparametern trainiert wurde. Eine andere mögliche Erklärung ist, dass der konkrete Sampling-Prozess ein anderer war. So ist es zum Beispiel möglich, dass für die Ergebnisse im Paper einzelne Punkte mehrfach decodet wurden, bis man ein valides Ergebnis erhält.(Dies wurde zumindest in anderen dargestellten Experimenten in ähnlicher Form getan.) Die Autoren machen in ihrem Paper deutlich, dass die Wahrscheinlichkeit valide Moleküle zu generieren deutlich steigt, wenn man nur die unmittelbare Nachbarschaft eines Punktes decodet. Aufgrund der niedrigen Validität sind die ermittelten Werte für den Anteil der neuartigen und einzigartigen generierten Moleküle fast bedeutungslos, da sie sich in diesem Fall auf 24 von 10.000 generierten SMILES-Strings beziehen. (Beide Werte sind gleich 100%.)

5.1.2 Fragment-based-VAE

Der für den Fragment-based-VAE ermittelte Anteil an validen Molekülen ist mit 99,99% fast optimal, zeigt aber auch, dass bei diesem Prozess durch die grundlegende Stochastizität noch immer nicht valide SMILES-Strings generiert werden können. Allerdings ist das bei einem von 10.000 generierten SMILES praktisch vernachlässigbar. Der prozentuale Anteil an neuartigen, generierten Molekülen ist mit 97,06% auch sehr gut, allerdings nicht optimal. Der prozentuale Anteil an einzigartigen, generierten Molekülen ist mit 12,78% nicht besonders gut. Dieser Wert zeigt auf, dass große Teile des Latent Spaces mit den selben Molekülen ausgefüllt werden. Des Weiteren

deutet dieser Wert daraufhin, dass Interpolation in diesem vermutlich nicht möglich sein wird.

5.1.3 CGVAE

Mit 100% validen generierten Molekülen ist der CGVAE in dieser Kategorie optimal. 99,97% der vom CGVAE generierten Ergebnisse sind einzigartig. Das bedeutet in diesem Zusammenhang, dass von 10.000 zufällig ausgewählten Punkten aus dem Latent Space nur 3 in Moleküle im Zinc-Datensatz enthalten waren. Zusammengenommen mit dem Umstand, dass 100,00% der generierten Moleküle, einzigartige Moleküle sind, ist dieses Ergebnis so unglaublich gut, dass es betreffend dieser drei Benchmarks pratkisch den Optimalfall darstellt.

5.2 Die Graphen

5.2.1 ChemVAE

Aufgrund der im vorigen Abschnitt bereits beschriebenen geringen Validität der vom betrachteten ChemVAE generierten SMILES-Strings geben die auf Basis dieser Daten ermittelten Eigeschaftsverteilungen und der durchschnittlichen Anzahlen der ermittelten Strukturelemente nur ein verzerrtes Bild der Originaldaten wieder. Die Verteilungen der QED-Werte und der SAS-Werte der ermittelten Daten wirken nicht so, als würden sie sich den Eigeschaftsverteilungen der Originaldaten besonders anpassen. Die LogP-Werte sind zufällig ähnlich denen der Originaldaten.

Die durchschnittlichen Anzahlen der in den generierten Molekülen enthaltenen Strukturelemente sind für den ChemVAE genauso nur auf 24 Molekülen ermittelt worden. Es fällt auf, dass die Moleküle im Durchschnitt fast alle weniger pro Molekül vorhanden sind. Dies liegt vermutlich daran, dass die generierten 24 validen Moleküle kürzer sind. Dies wiederrum ist vermutlich darin begründet, dass gilt je länger ein generierter SMILES-String ist, desto höher auch die Wahrscheinlichkeit, dass ein Fehler in diesem vorliegt.

5.2.2 Fragment-based-VAE

Die generierten Eigenschaftsverteilungen passen sich optisch an die Eigenschaftsverteilungen der Originaldaten an. Natürlich bekommt man keine perfekte Überlagerung, aber dabei wird ein Datensatz, bestehend aus circa 250.000 Molekülen, mit einer Stichprobe aus 10.000 Molekülen

verglichen. Die vorliegenden graphischen Ergebnisse deuten insgesamt darauf hin, dass der Fragment-based-VAE tatsächlich lernt, eine Verteilung von Daten zu generieren, welche der der Originaldaten ähnelt.

Ähnliches gilt an dieser Stelle auch für die Darstellungen der durchschnittlichen Anzahlen der enthaltenden strukturellen Eigenschaften.

5.2.3 CGVAE

Die Eigenschaftsverteilungen, welche der CGVAE generiert sind für die LogP- und die QED-Werte ähnlich denen der Originaldaten. Für den Synthetic Accessibility-Score(SAS) jedoch unterscheidet sich die Verteilung ein wenig. Man könnte an dieser Stelle argumentieren, dass das ein leichtes Manko ist, allerdings ist das in Anbetracht dessen, dass es sich hier nur um eine Stichprobe handelt und alle anderen Werte des CGVAE praktisch optimal sind, eine Nichtigkeit.

Für die durchschnittlichen, strukturellen Anzahlen der Struktureigenschaften wird, ähnlich wie beim Fragment-based-VAE deutlich, dass die generierten Moleküle in diesen Punkten ähnlich den Originaldaten sind.

6 Fazit

Was den Sampling-Prozess des ChemVAE angeht, so wäre es durchaus interessant zu erfahren, wie die Autoren im Detail vorgegangen sind. Denn obwohl Graph-basierte-Models normalerweise bezüglich der wichtigsten Benchmarks zeichenweise Language Models übertreffen, so ist dennoch eine deutlich bessere Performance mit zeichenweisen Language Models möglich. Podda u. a. (2020) Dennoch zeigen diese Ergebnisse zusammen mit dem Wissen aus den betrachteten Papers, dass der Einsatz von zeichenweisen Language Models in ihrer Grundform erstmal nicht optimal für den Einsatz als VAEs, welche SMILES lernen, sind.

Was den Fragment-basierten Ansatz angeht, so ist es interessant zu sehen, wie man die Performance eines VAEs deutlich verbessern kann, indem man die größten Schwächen von SMILES durch den Einsatz von Fragmenten praktisch effektiv maskiert. Dadurch sind Klammerfehler und das Öffnen und Schließen von Ringen im Molekül nichts mehr, was der variationale Autoencoder explizit lernen muss. Leider hat dieser Ansatz noch nicht den Anteil an einzigartigen Molekülen, um den eigenen Latent Space effektiv ausnutzen zu können. Möglicherweise könnte ein weiterer Regularisierungsterm in der Loss-Funktion helfen oder das stärkere Gewichten des KL-Divergenz Terms in der Loss-Funktion(β-VAE).

Die Architektur des CGVAE ist schon dadurch beeindruckend, dass mehrere neuronale Netzwerke im Decoder genutzt werden. Die Ergebnisse machen eindeutig klar, dass der CGVAE sich für das Lernen von Molekülen vortrefflich eignet. Gerade deswegen wäre es interessant diesen VAE genauer zu erforschen, und versuchen rauszufinden, wo dieser möglicherweise noch Schwächen hat.

Insgesamt sind noch weitere Untersuchungen über den Latent Space und das Verhalten des generativen Models von variationalen Autoencodern interessant. So könnte man zum Beispiel das Interpolationsverhalten der variationalen Autoencoder betrachten. Des Weiteren ist das genauere Betrachten von verschiedenen konzeptionellen und architekturellen Änderungen interessant und welche Auswirkungen diese auf die Performance der variationalen Autoenco-

der haben. Darüber hinaus gibt es noch viele andere interessante variationale Autoencoder Architekturen, bei denen es unter verschiedenen Gesichtspunkten Sinn machen würde diese zu untersuchen. Darüber hinaus ist es natürlich auch interessant, die Anwendbarkeit von diesen und ähnlichen variationalen Autoencoder und den von ihnen generierten Molekülrepräsentationen als Grundlage für die Wirkstoffentwicklung detaillierter zu untersuchen.

Literaturverzeichnis

[Aggarwal 2018] Aggarwal, Charu C.: *Neural Networks and Deep Learning - A Textbook*. Berlin, Heidelberg : Springer, 2018. – ISBN 978-3-319-94463-0

[Amézqueta u. a. 2020] Amézqueta, S. ; Subirats, X. ; Fuguet, E. ; Rosés, M. ; Ràfols, C.: Chapter 6 - Octanol-Water Partition Constant. In: Poole, Colin F. (Hrsg.): *Liquid-Phase Extraction*. Elsevier, 2020 (Handbooks in Separation Science), S. 183–208. – URL https://www.sciencedirect.com/science/article/pii/B9780128169117000062. – ISBN 978-0-12-816911-7

[Bickerton u. a. 2012] Bickerton, Richard ; Paolini, Gaia ; Besnard, Jérémy ; Muresan, Sorel ; Hopkins, Andrew: Quantifying the chemical beauty of drugs. In: *Nature chemistry* 4 (2012), 02, S. 90–8

[Britz 2015] Britz, Denny: *Understanding Convolutional Neural Networks for NLP – WildML*. November 2015. – URL http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/. – Zugriffsdatum: 2021-09-19

[Chen u. a. 2020] Chen, Chien-Hua ; Lin, Po-Hsiang ; Hsieh, Jer-Guang ; Cheng, Shu-Ling ; Jeng, Jyh-Horng: Robust Multi-Class Classification Using Linearly Scored Categorical Cross-Entropy. In: *2020 3rd IEEE International Conference on Knowledge Innovation and Invention (ICKII)*, August 2020, S. 200–203

[Cho u. a. 2014] Cho, Kyunghyun ; Merrienboer, Bart van ; Gülçehre, Çaglar ; Bougares, Fethi ; Schwenk, Holger ; Bengio, Yoshua: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In: *CoRR* abs/1406.1078 (2014). – URL http://arxiv.org/abs/1406.1078

[Chollet u. a. 2015] Chollet, Francois u. a.: *Keras*. 2015. – URL https://github.com/fchollet/keras

[David u. a. 2020] David, Laurianne ; Thakkar, Amol ; Mercado, Rocío ; Engkvist, Ola: Molecular representations in AI-driven drug discovery: a review and practical guide. In: *Journal of Cheminformatics* 12 (2020), September, S. 56. – URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7495975/. – Zugriffsdatum: 2021-08-16. – ISSN 1758-2946

[Degen u. a. 2008] Degen, Jörg ; Wegscheid-Gerlach, Christof ; Zaliani, Andrea ; Rarey, Matthias: On the Art of Compiling and Using 'Drug-Like' Chemical Fragment Spaces. In: *ChemMedChem* 3 (2008), 10, S. 1503–7

[Erdogan 2017] Erdogan, Goker: *Variational Autoencoder Explained*. August 2017. – URL http://gokererdogan.github.io/2017/08/15/variational-autoencoder-explained/. – Zugriffsdatum: 2021-09-21

[Ertl und Schuffenhauer 2009] Ertl, P ; Schuffenhauer, Ansgar: Estimation of Synthetic Accessibility Score of Drug-Like Molecules Based on Molecular Complexity and Fragment Contributions. In: *Journal of cheminformatics* 1 (2009), 06, S. 8. ISBN 978-1-926692-62-3

[Gilmer u. a. 2017] Gilmer, Justin ; Schoenholz, Samuel S. ; Riley, Patrick F. ; Vinyals, Oriol ; Dahl, George E.: Neural Message Passing for Quantum Chemistry. In: *CoRR* abs/1704.01212 (2017). – URL http://arxiv.org/abs/1704.01212

[Gómez-Bombarelli u. a. 2018] Gómez-Bombarelli, Rafael ; Wei, Jennifer N. ; Duvenaud, David ; Hernández-Lobato, José Miguel ; Sánchez-Lengeling, Benjamín ; Sheberla, Dennis ; Aguilera-Iparraguirre, Jorge ; Hirzel, Timothy D. ; Adams, Ryan P. ; Aspuru-Guzik, Alán: Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. In: *ACS Central Science* 4 (2018), Januar, Nr. 2, S. 268–276. – URL https://doi.org/10.1021/acscentsci.7b00572

[Goodfellow u. a. 2016] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016. – http://www.deeplearningbook.org

[Hamilton 2020] Hamilton, William L.: Graph Representation Learning. In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14 (2020), Nr. 3, S. 1–159

[Heller u. a. 2013] Heller, Stephen ; McNaught, Alan ; Stein, Stephen ; Tchekhovskoi, Dmitrii ; Pletnev, Igor: InChI - The worldwide chemical structure identifier standard. In: *Journal of cheminformatics* 5 (2013), 01, S. 7

[Hochreiter und Schmidhuber 1997] Hochreiter, Sepp ; Schmidhuber, Jürgen: Long Short-Term Memory. In: *Neural Computation* 9 (1997), November, Nr. 8, S. 1735–1780. – URL https://doi.org/10.1162/neco.1997.9.8.1735

[Jordan 2018] Jordan, Jeremy: *Variational autoencoders*. März 2018. – URL https://www.jeremyjordan.me/variational-autoencoders/. – Zugriffsdatum: 2021-09-21

[Kalchbrenner u. a. 2014] Kalchbrenner, Nal ; Grefenstette, Edward ; Blunsom, Phil: A Convolutional Neural Network for Modelling Sentences. (2014)

[Kelly u. a. 2021] Kelly, Sean ; Lupini, Andrea ; Epureanu, Bogdan: Data-Driven Modeling Approach for Mistuned Cyclic Structures. In: *AIAA Journal* 59 (2021), 04, S. 1–13

[Kingma und Welling 2014] Kingma, Diederik P. ; Welling, Max: Auto-Encoding Variational Bayes. (2014)

[Kingma und Welling 2019] Kingma, Diederik P. ; Welling, Max: An Introduction to Variational Autoencoders. In: *CoRR* abs/1906.02691 (2019). – URL http://arxiv.org/abs/1906.02691

[Kraev 2018] Kraev, Egor: Grammars and reinforcement learning for molecule optimization. In: *arXiv:1811.11222 [physics, stat]* (2018), November. – URL http://arxiv.org/abs/1811.11222. – Zugriffsdatum: 2021-08-16. – arXiv: 1811.11222

[Kubat 2017] Kubat, Miroslav ; Kubat, Miroslav (Hrsg.): *An Introduction to Machine Learning*. Cham : Springer International Publishing, 2017. – 251–271 S. – URL https://doi.org/10.1007/978-3-319-63913-0_13. – Zugriffsdatum: 2021-08-18. – ISBN 978-3-319-63913-0

[Landrum u. a. 2020] Landrum, Greg ; Tosco, Paolo ; Kelley, Brian ; sriniker ; gedeck ; NadineSchneider ; Vianello, Riccardo ; Ric ; Dalke, Andrew ; Cole, Brian ; Alexander-Savelyev ; Swain, Matt ; Turk, Samo ; N, Dan ; Vaucher, Alain ; Kawashima, Eisuke ; Wójcikowski, Maciej ; Probst, Daniel ; godin guillaume ; Cosgrove, David ; Pahl, Axel ; JP ; Berenger, Francois ; strets123 ; JLVarjo ; O'Boyle, Noel ; Fuller, Patrick ; Jensen, Jan H. ; Sforna, Gianluca ; DoliathGavid: *rdkit/rdkit: 2020_03_1 (Q1 2020) Release*. März 2020. – URL https://doi.org/10.5281/zenodo.3732262. – Zugriffsdatum: 2021-08-16

[Le und Mikolov 2014] Le, Quoc V. ; Mikolov, Tomás: Distributed Representations of Sentences and Documents. In: *CoRR* abs/1405.4053 (2014). – URL http://arxiv.org/abs/1405.4053

[Li u. a. 2016] Li, Yujia ; Tarlow, Daniel ; Brockschmidt, Marc ; Zemel, R.: Gated Graph Sequence Neural Networks. In: *CoRR* abs/1511.05493 (2016)

[Liu u. a. 2018] Liu, Qi ; Allamanis, Miltiadis ; Brockschmidt, Marc ; Gaunt, Alexander L.: Constrained Graph Variational Autoencoders for Molecule Design. In: *The Thirty-second Conference on Neural Information Processing Systems* (2018)

[McNaught und Wilkinson 2014] McNaught, A. D. ; Wilkinson, A. ; Gold, Victor (Hrsg.): *The IUPAC Compendium of Chemical Terminology – The Gold Book*. 2. International Union of Pure and Applied Chemistry (IUPAC), 2014. – URL https://doi.org/10.1351/goldbook. – ISBN 0-9678550-9-8

[Minkin 1999] Minkin, V. I.: Glossary of terms used in theoretical organic chemistry:. In: *Pure and Applied Chemistry* 71 (1999), Nr. 10, S. 1919–1981. – URL https://doi.org/10.1351/pac199971101919

[Moldoveanu und David 2015] Moldoveanu, Serban ; David, Victor: Chapter 5 - Phase Transfer in Sample Preparation. In: Moldoveanu, Serban (Hrsg.) ; David, Victor (Hrsg.): *Modern Sample Preparation for Chromatography*. Amsterdam : Elsevier, 2015, S. 105–130. – URL https://www.sciencedirect.com/science/article/pii/B9780444543196000050. – ISBN 978-0-444-54319-6

[Mortimer und Müller 2020] Mortimer, Charles E. ; Müller, Ulrich: *Chemie - Das Basiswissen der Chemie*. Stuttgart : Georg Thieme Verlag, 2020. – ISBN 978-3-132-42275-9

[Mouchlis u. a. 2021a] Mouchlis, Varnavas D. ; Afantitis, Antreas ; Serra, Angela ; Fratello, Michele ; Papadiamantis, Anastasios G. ; Aidinis, Vassilis ; Lynch, Iseult ; Greco, Dario ; Melagraki, Georgia: Advances in De Novo Drug Design: From Conventional to Machine Learning Methods. In: *International Journal of Molecular Sciences* 22 (2021), Nr. 4. – URL https://www.mdpi.com/1422-0067/22/4/1676. – ISSN 1422-0067

[Mouchlis u. a. 2021b] Mouchlis, Varnavas D. ; Afantitis, Antreas ; Serra, Angela ; Fratello, Michele ; Papadiamantis, Anastasios G. ; Aidinis, Vassilis ; Lynch, Iseult ; Greco, Dario ; Melagraki, Georgia: Advances in De Novo Drug Design: From Conventional to

Machine Learning Methods. In: *International Journal of Molecular Sciences* 22 (2021), Februar, Nr. 4, S. 1676. – URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7915729/. – Zugriffsdatum: 2021-08-16. – ISSN 1422-0067

[Nakkiran 2020] Nakkiran, Preetum: Learning Rate Annealing Can Provably Help Generalization, Even for Convex Problems. In: *arXiv:2005.07360 [cs, stat]* (2020), Mai. – URL http://arxiv.org/abs/2005.07360. – Zugriffsdatum: 2021-09-02. – arXiv: 2005.07360

[Nielsen 2015] Nielsen, Michael A.: *Neural Networks and Deep Learning*. URL http://neuralnetworksanddeeplearning.com. – Zugriffsdatum: 2021-08-06, 2015. – http://neuralnetworksanddeeplearning.com

[Olah 2015] Olah, Christopher: *Understanding LSTM Networks*. August 2015. – URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/. – Zugriffsdatum: 2021-08-06. – https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[Podda u. a. 2020] Podda, Marco ; Bacciu, Davide ; Micheli, Alessio: A Deep Generative Model for Fragment-Based Molecule Generation. In: Chiappa, Silvia (Hrsg.) ; Calandra, Roberto (Hrsg.): *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics* Bd. 108, PMLR, 26–28 Aug 2020, S. 2240–2250. – URL http://proceedings.mlr.press/v108/podda20a.html

[Ramakrishnan u. a. 2014] Ramakrishnan, Raghunathan ; Dral, Pavlo ; Rupp, Matthias ; Lilienfeld, Anatole von: Quantum chemistry structures and properties of 134 kilo molecules. In: *Scientific Data* 1 (2014), 08

[Saldívar-González u. a. 2020] Saldívar-González, Fernanda I. ; Huerta-García, C. S. ; Medina-Franco, José L.: Chemoinformatics-based enumeration of chemical libraries: a tutorial. In: *Journal of Cheminformatics* 12 (2020), Oktober, Nr. 1, S. 64. – URL https://doi.org/10.1186/s13321-020-00466-z. – Zugriffsdatum: 2021-09-20. – ISSN 1758-2946

[Shafkat 2018] Shafkat, Irhum: *Intuitively Understanding Variational Autoencoders*. April 2018. – URL https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf. – Zugriffsdatum: 2021-09-05

[Vollhardt und Schore 2020] Vollhardt, Kurt Peter C. ; Schore, Neil E.: *Organische Chemie* -. 6. Weinheim : Wiley-VCH Verlag GmbH, 2020. – ISBN 978-3-527-34582-3

[Watt und Du Plessis 2020] Watt, Nathan ; Du Plessis, Mathys: Towards robot vision using deep neural networks in evolutionary robotics. In: *Evolutionary Intelligence* (2020), 10

[Wei u. a. 2021] Wei, Jennifer N. ; Sánchez-Lengeling, Benjamín ; Sheberla, Dennis ; Gómez-Bombarelli, Rafael ; Aspuru-Guzik, Alán: *aspuru-guzik-group/chemical_vae*. August 2021. – URL https://github.com/aspuru-guzik-group/chemical_vae. – Zugriffsdatum: 2021-08-08. – original-date: 2017-12-20T21:41:33Z

[Weininger 1988] Weininger, David: SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. In: *Journal of Chemical Information and Computer Sciences* 28 (1988), Februar, Nr. 1, S. 31–36. – URL https://pubs.acs.org/doi/abs/10.1021/ci00057a005. – Zugriffsdatum: 2021-08-16. – Publisher: American Chemical Society. – ISSN 0095-2338

[Weng 2018] Weng, Lilian: From Autoencoder to Beta-VAE. In: *lilianweng.github.io/lil-log* (2018). – URL http://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

[Zhang und Wallace 2016] Zhang, Ye ; Wallace, Byron: A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. (2016)

[Zhou u. a. 2020] Zhou, Jie ; Cui, Ganqu ; Hu, Shengding ; Zhang, Zhengyan ; Yang, Cheng ; Liu, Zhiyuan ; Wang, Lifeng ; Li, Changcheng ; Sun, Maosong: Graph neural networks: A review of methods and applications. In: *AI Open* 1 (2020), S. 57–81. – URL https://doi.org/10.1016/j.aiopen.2021.01.001

Glossar

- **Chemie**: „Chemie ist die Wissenschaft, die sich mit der Charakterisierung, Zusammensetzung und Umwandlung von Stoffen befasst.“ – (Mortimer und Müller, 2020, Seite 19)

- **Atom**: Ein Atom ist das kleinste Teilchen, aus welchen ein Element aufgebaut ist. (Mortimer und Müller, 2020, Seite 32) *Elemente* sind die Stoffe, die nicht weiter in einfacheren Stoffe zerlegt werden können, aus ihnen bestehen alle anderen Stoffe. Jedes Element hat einen Elementnamen und ein chemisches Symbol. (Mortimer und Müller, 2020, Seite 19) Das Atom wiederum ist aus kleineren Teilchen aufgebaut(*Neutronen*, *Protonen* und *Elektronen*). Die Ordnungszahl eines Elements entspricht der Zahl der Protonen in seinen Atomen. (Mortimer und Müller, 2020, Seite 32) Man unterscheidet Elemente an ihrer Anzahl der Protonen(deswegen der Term Ordnungszahl).McNaught und Wilkinson (2014); Minkin (1999) Die chemischen Eigenschaften eines Atoms sind vor allem davon abhängig, um welches Element es sich handelt, dass bedeutet also von der Anzahl der Protonen, die es enhält. (Mortimer und Müller, 2020, Seite 37)

Einige Elemente beispielhaft, mit ihrem chemischen Symbol, und ihrer Ordnungszahl:

Name des Elements	**chemisches Symbol**	**Ordnungszahl**
Wasserstoff	H	1
Sauerstoff	O	8
Kohlenstoff	C	6
Stickstoff	N	7
Schwefel	S	16
Chlor	Cl	17
Brom	Br	35

Tabelle 1: Einige Elemente mit chemischem Symbol und Ordnungszahl.

Mortimer und Müller (2020)

- **chemische Bindung(en)**: „*Nichtmetall-Atome* werden durch *kovalente Bindungen* zu *Molekülen* verknüpft. In einer kovalenten *Einfachbindung* haben zwei Atome ein gemeinsames Elektronenpaar. In einer *Doppelbindung* sind es zwei, in einer *Dreifachbindung* drei gemeinsame Elektronenpaare.“ – (Mortimer und Müller, 2020, Seite 117)
- **Kovalente Bindung**: Eine Bindungstyp, zwischen zwei Atomen, bei denen sich die Atome Elektronen teilen. (Mortimer und Müller, 2020, Seite 580)
- **Molekül**: Atome, die durch Bindungen zu einem Teilchen verbunden sind, bezeichnet man als Moleküle.(Mortimer und Müller, 2020, Seite 682)
- **Organische Chemie**: Die Chemie der Verbindungen des Kohlenstoffs, mit einigen wenigen Ausnahmen. Der Begriff *organisch* stammt aus einer Zeit, als man glaubte, nur lebendige Wesen, wie Tiere und Pflanzen könnten diese Stoffe erzeugen. (Mortimer und Müller, 2020, Seite 22)
- **Valenz**: Die Anzahl der einbindigen Bindungspartner, mit denen ein betrachtetes Element Bindungen eingehen kann. McNaught und Wilkinson (2014)
- **Isomere**: Verbindungen mit der selben Summenformel, aber unterschiedlichen Strukturen. Isomere haben dadurch auch unterschiedliche physikochemische Eigenschaften, verhalten sich also wie verschiedene Moleküle. (Mortimer und Müller, 2020, Seite 678)
- **Stereochemie**: In diesem Fall die organische Stereochemie, ist der Teilbereich der Chemie, der sich mit der räumlichen, dreidimensionalen Darstellung von Molekülen befasst. (Mortimer und Müller, 2020, Seite 544)
- **Konstitutionsisomere**: auch *Stereoisomere*; bezeichnen Isomere, bei denen die selben Atome, miteinander durch Bindungen verknüpft sind, die sich aber in ihrer dreidimensionalen Anordnung unterscheiden. (Mortimer und Müller, 2020, Seite 544 bis 545)
- **chiral/Chiralität**: Moleküle von denen zwei gleiche, aber spiegelbildliche Formen exisitieren, werden als chiral bezeichnet. Man bezeichnet diese beiden Formen als *Enantiomere*. In ihren physikalischen Eigenschaften unterscheiden sich Enantiomere nur in ihrem Verhalten gegenüber polarisiertem Licht.(Mortimer und Müller, 2020, Seite 487)

  Chiralität bzw. Enantiomere sind in diesem Zusammenhang wichtig, weil es anfänglich nicht möglich war verschiedene Enatiomere eines Moleküls darzustellen.

- **Aromatizität**: Beschreibt den Umstand und die räumlichen, und Ladungseigenschaften, zyklischer Molekülstrukturen, welche innerhalb ihres/ihrer Ringstrukturen Elektronendelokalisation aufweisen, welche für ihre relative höhere Stabilität verantwortlich ist. McNaught und Wilkinson (2014)

- **Wirkstoff-Design**: Rationales Wirkstoff-Design, bezeichnet den Einsatz ausgeklügelter Software, um bei der Entwicklung neuer Wirkstoffe zu helfen. (Vollhardt und Schore, 2020, Seite 1338)

- **Synthese/synthetisieren**: Als Synthese bezeichnet man die Herstellung von Verbindungen/Molekülen. (Vollhardt und Schore, 2020, Seite 3)

- **Lipophilizität**: Ein Molekül gilt dann als *lipophil*, wenn es sich besonders gut in Fettähnlichen Lösungsmitteln, lösen lässt. McNaught und Wilkinson (2014)

- **Stoffmengenkonzentration**: Eine Konzentration generell ist die Menge, eines Stoffes, welcher in einer Lösung, gelöst ist. Die Stoffmengenkonzentration(notiert als c) sagt aus, wie viel einer gegeben Stoffmenge eines Stoffes, pro Volumen, in der betrachteten Lösung gelöst ist. Sie wird in der Einheit mol pro Liter($\frac{mol}{L}$) angegeben.(Mortimer und Müller, 2020, Seite 56)

$$c = \frac{n}{V} \tag{1}$$

(Mortimer und Müller, 2020, Seite 56)

- **Mol**: Einheit der Stoffmenge; 1 mol = Stoffmenge, die aus N_A Teilchen besteht; $N_A = 6,02214076 \cdot 10^{23}$ (N_A ist die Avogadro-Konstante.) (Mortimer und Müller, 2020, Seite 44)

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 05. Oktober 2021 Darius Szablowski