MolGrad: Moleküle generieren und optimieren mit KI

Paul A. M. Wollenhaupt
paul.wollenhaupt@gmail.com
Gymnasium Bad Zwischenahn-Edewecht

Zusammenfassung

Das Entwickeln von Medikamenten ist teuer und dauert lange, weil es schwierig ist, unter den rund 10^{60} wirkstoffähnlichen Molekülen die richtigen zu finden. Künstliche Intelligenz kann Muster aus großen Datensätzen lernen und hat das Potential die Wirkstoffentwicklung effizienter zu gestalten. Konventionelle Methoden des tiefen Lernens sind allerdings schwierig auf Moleküle anzuwenden, weil sie diskret und die Atome nicht nummeriert sind. MolGrad versucht diese Probleme zu lösen, indem es Überlagerungen von Molekülen modelliert und gehört damit zur vielversprechenden Gruppe der Score-basierten generativen Modelle. Die Überlagerungen werden durch einen neuen, interpretierbaren Diffusionsprozess erzeugt und für das Modell wurde eine neuartige neuronale Architektur entworfen, die Transformer mit neuronalen Netzen für Graphen vereint. Gezeigt wird, dass mit MolGrad Moleküle nicht nur erfolgreich generiert werden können, sondern auch, dass Bearbeiten und Optimieren natürliche Vorteile der Methode sind.

Inhaltsverzeichnis

1 Einleitung und Leitfragen

Eine neuartige Methode, die mit Hilfe von künstlicher Intelligenz, Moleküle generiert und nach gewünschten Eigenschaften optimiert und somit Potential hat in der Medikamentenentwicklung wichtige Aufgaben effizienter zu gestalten, soll in dieser Arbeit vorgestellt werden. Dafür werden zuerst das Wirkstoffdesign und die Rolle von maschinellem Lernen erläutert. Die übrige Arbeit orientiert sich an den folgenden Leitfragen:

1. Wie funktionieren aktuelle Methoden zum Generieren von Molekülen?
2. Was sind deren Vor- und Nachteile, welche Features sind beim Wirkstoffdesign gewünscht?
3. Können vorteilhafte Eigenschaften auch durch andere Methoden erreicht werden?
4. Wie kann Score-basiertes Modellieren für die Medikamentenentwicklung genutzt/abgewandelt werden?
5. Wie sieht eine dazu passende, neue neuronale Architektur aus?
6. Wie weit kann die Funktionsfähigkeit der neuen Methode beim Generieren, Bearbeiten und Optimieren von synthetischen Daten und Molekülen demonstriert werden?
7. Was sind die Grenzen der Methode?

2 Wirkstoffdesign

Wirkstoffdesign bedeutet, nach neuen Medikamenten zu forschen. Wie in Abbildung 1 zu sehen ist, besteht dieser Prozess aus mehreren Stufen, in denen mögliche Wirkstoffe immer strengeren Tests unterzogen werden.

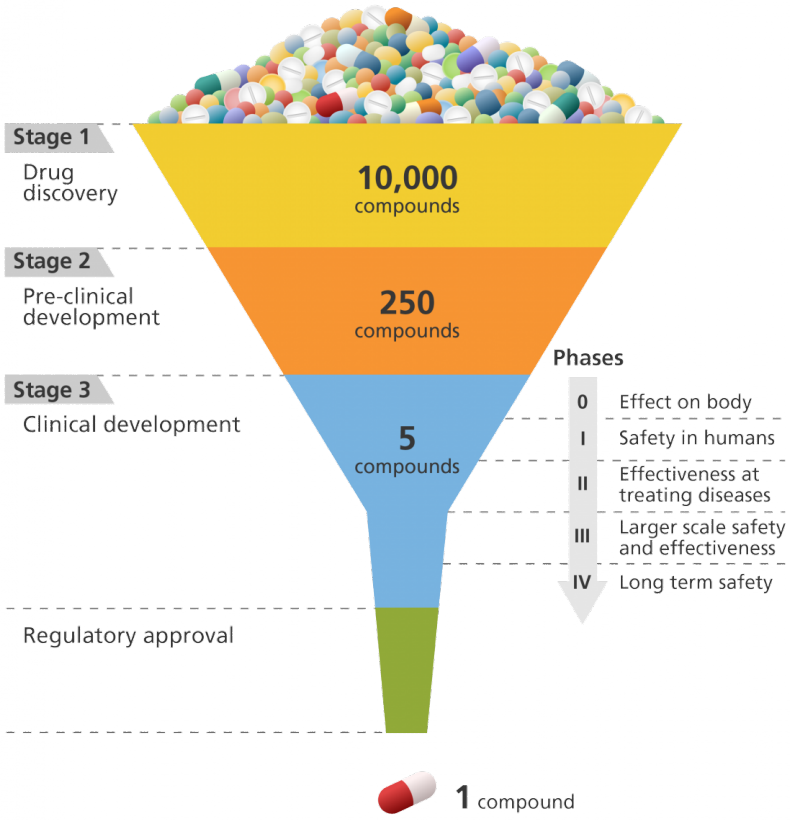

Abbildung 1: Menge der potenziellen Wirkstoffe im Designverlauf bis zur Genehmigung eines Medikaments

Deshalb muss die Anzahl der Moleküle am Anfang entsprechend hoch sein, um zum Schluss ein Medikament auf den Markt bringen zu können, welches durchschnittlich etwa 1,07 Milliarden Euro kostet [66]. 30 % dieser Kosten fallen in einem Zeitraum von etwa fünf Jahren in den ersten Stufen bei einfachen Tests von vielen Molekülen an [17]. Die Zahl der zu testenden Moleküle kann dort durch clevere Algorithmen reduziert werden. In späteren Schritten mit Menschenversuchen ist das wegen hoher Sicherheitsstandards nicht möglich [17].

2.1 Weshalb ist maschinelles Lernen nötig?

Die Menge der wirkstoffähnlichen Moleküle überschreitet 10^{60} deutlich [9]. Selbst Hochdurchsatz-Screening mit einem Durchsatz von etwa 10^4 Verbindungen pro Woche reicht deswegen nicht aus, um naiv nach Molekülen mit gewünschten Eigenschaften zu suchen [47].
Menschen haben deswegen lange die Aufgabe des rationalen Wirkstoffdesigns übernommen. Dabei geht es darum, kleine Moleküle mit gewünschten Eigenschaften zu entwerfen, sodass sie auf ein biologisches Molekül bekannter oder unbekannter Struktur passen [48]. Angesichts der überwältigenden Menge an möglichen Molekülen und einer generell geringen Erfolgsquote [40] eignet sich ein durch Computer automatisierter Ansatz, bei dem Maschinelles Lernen (ML) in doppelter Hinsicht hilft: Anhand von großen Datensätzen kann ein Modell zum Generieren und Modifizieren von Molekülen erlernt werden oder die Interaktion zwischen Molekülen und biologischen Zielen wird prognostiziert [19].

3 Generatives Modellieren von Molekülen

Im Folgenden sollen die wichtigsten ML-basierten Ansätze zum generativen Modellieren von Molekülen vorgestellt und diskutiert werden. Voraussetzung für moderne ML-Systeme sind tiefe künstliche neuronale Netze (KNN), die aus aneinandergeketteten, parametrischen, differenzierbaren Funktionen bestehen [25], deren Parameter θ effizient optimiert werden können [55], um arbiträre, kontinuierliche Funktionen zu approximieren [42].

3.1 Moleküle

Bevor die generativen Methoden näher betrachtet werden, ist es sinnvoll, näher darauf einzugehen, was Moleküle im Kontext von ML ausmachen, und wie sie dargestellt werden können.
Moleküle sind ungerichtete Graphen, bei denen die Knoten Atome und die Kanten Bindungen darstellen. Atome werden einem Element zugeordnet und Bindungen haben eine Ordnung, die durch die Elektronenanzahl bestimmt ist[1] [16].
Moleküle müssen allerdings nicht als Graph dargestellt werden. Mit der „Simplified Molecular Input Line Entry Specification" (SMILES) können Moleküle auch als eine Reihe von Buchstaben und Zeichen abgebildet werden [5]. Aufgrund der Lesbarkeit und der linearen Form ist es oft einfach, mit der SMILES-Repräsentation zu arbeiten, obwohl ein bestimmtes Molekül durch mehrere SMILES-Strings beschrieben werden kann [16].
Im Rahmen von ML ist es außerdem wichtig, zu betrachten, dass Moleküle diskrete Objekte sind, da Atome und Bindungen entweder existieren oder nicht. Darüber hinaus können die Bindungen nur eine diskrete Zahl an Elektronen und die Atome nur eine diskrete Ordnungszahl haben.

3.2 Variationsautoencoder (VAE)

Autoencoder (AE) bestehen aus zwei KNN. Einem Encoder E, der einen hochdimensionalen Input auf eine versteckte Repräsentation $c = E_\theta(x)$ niedrigerer Dimension komprimiert und einem Decoder $\hat{x} = D_\theta(c)$, der den Input anhand dieser Repräsentation rekonstruieren soll [6].
AEs können allerdings keine neuen Daten generieren, da die Verteilung im versteckten Raum unbekannt ist. VAEs definieren diese Verteilung explizit und der Encoder kann dann nur die Parameter jener Verteilung bestimmen. Differenzierbar wird das ganze durch den Reparametrisierungstrick, bei dem erst eine Stichprobe mit Standardparametern entnommen wird, die dann umgerechnet wird [39].

$$\mathcal{L}(\theta) \triangleq \mathbb{E}_{x\sim p_{\text{data}}(x), z\sim\mathcal{N}(0,\text{I})}\left[\|x - \text{VAE}_\theta(x)\|_2^2 + \text{KL}(c, z)\right] \tag{1}$$

[1] Delokalisierte Bindungen oder Ladungen werden durch eine mesomere Grenzstruktur dargestellt und die Konformations- und Konfigurationsisomerien werden vernachlässigt.

Zu dem Rekonstruktionsterm in der Fehlerfunktion des VAEs wird noch die Kullback-Leibler-Divergenz addiert, damit die Parameter im versteckten Raum nicht zu weit von den Standardparametern abweichen und somit die generierten Beispiele besser zu den Daten passen. Die Fehlerfunktion in Gleichung 1 hat eine Normalverteilung als Prior, die KL-Divergenz kann deshalb auch direkt aus der Ausgabe des Encoders errechnet werden [39].
Das VAE-Modell aus [2] erzeugt eine Repräsentation für mehrere SMILES-Strings eines Moleküls und schränkt die möglichen Rekonstruktionen auf verschiedenen Wegen ein, um bessere Ergebnisse zu erzielen.

3.3 Erzeugende gegnerische Netzwerke (GANs)

GANs modellieren die Verteilung der Daten nur implizit und bestehen aus zwei künstlichen neuronalen Netzwerken (KNN), dem Generator G_ϑ und dem Diskriminator D_θ. Der Generator versucht, aus Rauschen $p_z(z)$ realistische Daten zu erzeugen und die Aufgabe des Diskriminators ist es, diese erzeugten Daten von den echten zu unterscheiden [23].
Da der Diskriminator differenzierbar ist, können beide Modelle mit einem Gradientenverfahren optimiert werden. Als Fehlerfunktion wird der Klassifizierungsfehler des Diskriminators benutzt. D_θ minimiert diesen, während G_ϑ versucht, diesen in die Höhe zu treiben. Es handelt sich also um ein Nullsummenspiel [23].

$$\min_{\vartheta} \max_{\theta} \mathbb{E}_{x\sim p(x),\, z\sim p_z(z)} \left[\log D_\theta(x) + \log(1 - D_\theta(G_\vartheta(z)))\right]. \tag{2}$$

Aktuelle Fortschritte in Form von besseren Netzwerk-Architekturen [36], modifizierten Kostenfunktionen [49], simplem Hochskalieren mit größeren Netzen, Batches und Datensätzen [10] haben dafür gesorgt, dass GANs beeindruckend realistische Bilder erzeugen können. Jedoch sind instabiles Training [53] und „Mode Collapse", ein Phänomen, bei dem nur ein Bruchteil der möglichen Datenpunkte generiert werden, sind immer noch offene Probleme [59].
Ohne weiteres können GANs nicht auf Moleküle angewendet werden, da für den Generator keine Gradienten $\nabla_x D_\theta(x)$ existieren können, wenn er diskrete Moleküle ausgibt. In MolGAN [12, 43] werden deswegen Methoden des bestärkenden Lernens genutzt, um die Gradienten indirekt zu schätzen und in [52] findet der Lernprozess des GANs im verborgenen Raum eines SMILES-Autoencoders statt, der kontinuierlich ist.

3.4 Tiefe autoregressive Modelle

Autoregressiv zu modellieren bedeutet, eine multivariate Wahrscheinlichkeitsverteilung in das Produkt der einzelnen bedingten Wahrscheinlichkeiten nach Gleichung 3 aufzuteilen [60]. Hier steht z für einen zufälligen Startwert und D für die Dimensionalität der Daten:

$$p(x) = \prod_{i=1}^{D} p(x_i \,|\, x_1, \ldots, x_{i-1}, z) \tag{3}$$

Ein tiefes KNN lernt dann anhand der vorherigen Werte, den nächsten vorherzusagen. Beim Generieren wird dann so getan, als wäre der vorhergesagte Wert der echte und der nächste wird vorhergesagt. Das Modell wird im Fall von diskreten Werten mit der Kreuzentropie als Fehlerfunktion trainiert [60]. Nachteilig ist jedoch, dass eine feste Reihenfolge nötig ist. Deswegen wird hier oft die SMILES-Repräsentation genutzt [51].

3.5 Flow Modelle

Flow Modelle nutzen invertierbare Funktionen, um Daten einer unbekannten Verteilung zu einer bekannten Verteilung „fließen" zu lassen. Für invertierbare Funktionen gilt das Substitutionsformular, mit dem die Veränderung der Wahrscheinlichkeit durch eine Funktion an einem Punkt durch die Determinante der Jacobi-Matrix nach Gleichung 4 berechnet werden kann [18].

$$p(f(x)) = \frac{p(x)}{\left|\det(\frac{\partial f(x)}{\partial x^T})\right|} \tag{4}$$

Anhand der bekannten Verteilung kann das Modell den Datenpunkten Wahrscheinlichkeiten zuordnen. Beim Training wird der Logarithmus dieser maximiert [18]. Generiert wird durch die bekannte Umkehrfunktion.

Invertierbare Funktionen können auch die Form von KNN annehmen und miteinander verkettet werden. Um das Berechnen der Jacobi-Determinanten effizient zu gestalten, wird auf affine Kupplungsschichten zurückgegriffen [38]. Ein weiterer Trick, der für das Modellieren von nicht ganz kontinuierlichen[2] Daten nötig ist, ist das Dequantisieren mit Rauschen [28].
Die Methode MoFlow nutzt die oben beschriebene Methode, um Moleküle in zwei Schritten zu generieren. Zuerst wird der Adjazenztensor erzeugt. Danach werden die Atome konditioniert auf diesen Tensor permutationsequivariant generiert [70].

3.6 Vergleich der Modellierungsmethoden

Die erläuterten generativen Methoden haben alle unterschiedliche Vorteile, weshalb sie sich für unterschiedliche Bereiche eignen. Wichtige Charakteristiken sind außerdem die induktiven Verzerrungen: Bei GANs wirken die generierten Daten realistisch auf D_θ, bei autoregressiven Modellen ist der jeweils vorausgehende Teil besonders wichtig für den kommenden, bei VAEs bedeuten leichte Abweichungen angesichts des Rekonstruktionsfehlers kleine Veränderungen in den Datenpunkten und in Flow-basierten Modellen stammen die Daten aus einem hochdimensionalen versteckten Raum mit einer bestimmten Verteilung. Zur Entwicklung von Wikstoffen passen allerdings keine dieser eingebauten Verzerrungen. Für die Wirkstoffentwicklung und das Arbeiten mit Molekülen sind zusätzlich noch die folgenden Eigenschaften wichtig:

	GAN	Autoregressiv	VAE	Flow
Abdeckung aller Modi	×	✓	✓	✓
Projizieren von Daten in den versteckten Raum	×	×	✓	✓
Permutationsinvarianz	×	×	×	×
Interpretierbarkeit des versteckten Raums	×	×	×	×
Freie Wahl der Architektur	×	✓	✓	∼

Die GAN-Architektur hat großen Einfluss auf die Stabilität des Trainings [54], und bei Flow-Modellen gibt es nur eine freie Wahl der Architektur innerhalb der Kupplungsschicht. In der Tabelle ist zu erkennen, dass GANs am schlechtesten in das Profil der Wirkstoffentwicklung fallen, während die VAE- und Flow-basierten Methoden die meisten gewünschten Eigenschaften abdecken.

4 MolGrad

Das Generieren, Bearbeiten und Optimieren von Molekülen mit Score-basierten Modellen ist noch nicht erforscht. MolGrad soll diese Lücke schließen und die Bezeichnung setzt sich dementsprechend aus „Mol" für Moleküle und „Grad" für den Gradienten der Score-Funktion zusammen.
Diffusionsmodelle sind angesichts der neuen, beeindruckenden Ergebnisse [29, 14, 58] und Anwendungsmöglichkeiten, die über das einfache Generieren hinausgehen [35, 33], sehr vielversprechend.
Diese Arbeit leistet zwei Beiträge: Ein neuer, robuster, interpretierbarer Diffusionsprozess für Graphen und eine passende neuronale Netzwerk-Architektur, die den Aufmerksamkeitsmechanismus aus Transformern [61] mit neuronalen Netzen für Graphen [24] vereint.

4.1 Langevin-Dynamik

Anstatt die normalisierte Verteilung $p(x)$ direkt abzuschätzen, ist es in vielen Fällen leichter und ausreichend, die Score-Funktion $\nabla_x \log p(x)$ zu modellieren. Durch Langevin-Dynamik, auch Langevin Sampling genannt, können nämlich Stichproben von einer Verteilung $p(x)$ entnommen werden, selbst wenn nur die Score-Funktion bekannt ist. Dazu muss Gleichung 5 wiederholt evaluiert werden. In jedem Schritt wird z aus einer Standard-Normalverteilung entnommen.

$$x_{t+1} = x_t + \alpha \nabla_x \log p(x_t) + \sqrt{2\alpha} z_{t+1} \tag{5}$$

Formel 5 stellt einen Wiener-Prozess mit der Scorefunktion als Drift dar. Der Drift sorgt, ähnlich wie bei dem Gradientenverfahren, dafür, dass sich x in Bereiche immer größerer Wahrscheinlichkeit begibt. Die Ergebnisse

[2] Bilder werden z. B. mit 8-bit Integern repräsentiert, weshalb $\mathbb{R}$ nur eine Annäherung ist.

der Langevin-Dynamik entsprechen genau der Ursprungsverteilung unter dem Limes von unendlich vielen Schritten t und einer unendlich kleinen Schrittgröße α [64].

4.1.1 Modellieren der Score-Funktion

Der einfachste Ansatz wäre, den empirischen Fehler zwischen dem KNN und der Scorefunktion wie in Gleichung 6 zu minimieren. Problematisch ist allerdings, dass die Score-Funktion dafür zumindest an einigen Punkten bekannt sein müsste.

$$\mathcal{L}(\theta) \triangleq \frac{1}{2}\,\mathbb{E}_{x\sim p(x)}\left[\|s_\theta(x) - \nabla_x \log p(x)\|_2^2\right] \tag{6}$$

$$= \frac{1}{2}\,\mathbb{E}_{x\sim p(x)}\left[\|s_\theta(x)\|_2^2 + 2\cdot \mathrm{tr}(\nabla_x s_\theta(x))\right] + C \tag{7}$$

Durch Integrieren der Score-Funktion kann eine von ihr unabhängige Fehlerfunktion erreicht werden (Gleichung 7) [31]. Das Evaluieren der Spur von $\nabla_x s_\theta(x)$ ist jedoch sehr rechenintensiv, weshalb es in der Praxis durch zufällige Projektionen approximiert werden muss [57].
Da sich empirisch zeigt, dass die meisten Datensätze auf einer geringer dimensionalen Mannigfaltigkeit liegen, also schlecht definiert ist, in welchem Bereich des Raums Wahrscheinlichkeiten von größer 0 zu erwarten sind, wird den Daten ein wenig normalverteiltes Rauschen hinzugefügt. Dies scheint außerdem auch das Training signifikant zu stabilisieren [56].
In [63] wurde festgestellt, dass normalverteiltes Rauschen die Gradienten in der verrauschten Verteilung in die Richtung der sauberen Datenpunkte $(x - \tilde{x})$ richtet. Um den Wert der Score-Funktion muss die Differenz noch durch die Varianz der Normalverteilung geteilt werden. Analog zu Gleichung 6 ergibt sich dann die folgende Fehlerfunktion:

$$\mathcal{L}(\theta,\sigma) \triangleq \frac{1}{2}\,\mathbb{E}_{x\sim p(x),\,\tilde{x}\sim\mathcal{N}(x,\sigma^2\mathrm{I})}\left[\left\|s_\theta(\tilde{x},\sigma) - \frac{x-\tilde{x}}{\sigma^2}\right\|_2^2\right] \tag{8}$$

In der Praxis hat sich dies als die effizienteste Fehlerfunktion herausgestellt und es wird oft mit unterschiedlichen Rauschstufen σ trainert [56], sodass die Schrittgröße und die Rauschstufe während der Langevin-Dynamik mit steigendem t verringert werden können. Dieser Algorithmus wird getemperte Langevin-Dynamik genannt [56].

4.2 Diffusionsmodelle und stochastische Differenzialgleichungen

Diffusionsmodelle, hier gleichbedeutend mit entrauschenden probabilistischen Diffusionsmodellen, betrachten das Generieren als Invertieren einer Markow-Kette, bei der den echten Daten graduell normalverteiltes Rauschen hinzugefügt wird. Sie wurden schon erfolgreich beim Generieren von Bildern und Audiodaten angewendet [29, 14]. Genauso wie die Langevin-Dynamik nutzen Diffusionsmodelle die Tatsache, dass die Summe von normalverteilten Zufallsvariablen auch normalverteilt ist, um den Vorwärtsprozess $\tilde{x}(t)$ effizient zu berechnen. Die Fehlerfunktion generalisiert[3] auch Gleichung 8:

$$\mathcal{L}(\theta,t) \triangleq \frac{1}{2}\,\mathbb{E}_{x_0\sim p(x),\,z\sim\mathcal{N}(0,\mathrm{I})}\left[\|s_\theta(\tilde{x}(x_0,z,t),t) + z\|_2^2\right] \tag{9}$$

Die Konzepte von der getemperten Langevin-Dynamik und Diffusionsmodellen wurden in [58] jedoch zur Diskretisierung von zwei Gruppen an stochastischen Differentialgleichungen generalisiert. Der Vorwärtsprozess dieser SDGL nimmt stets die folgende Form mit t von 0 bis 1 an:

$$\mathrm{d}x = f(x,t)\mathrm{d}t + g(t)\mathrm{d}w \tag{10}$$

Gleichung 11 beschreibt die getemperte Langevin-Dynamik und mit Formel 12 wird der Diffusionsprozess beschrieben. Langevin-Dynamik gehört zur Klasse VE, Prozesse bei denen die Varianz stark steigen muss, um die Daten zu übertönen. Der Diffusionsprozess gehört der Klasse VE an, da bei ihm mit dem Rauschen

[3] Es könnte noch weiter generalisiert werden, da z. B. in [14] die Manhattan-Distanz genutzt wurde.

interpoliert wird, weshalb die Varianz nicht die Größenordnung ändert. $\sigma^2(t)$ und $\beta(t)$ sind Hyperparameter.

$$\mathrm{d}x = \sqrt{\frac{\mathrm{d}\sigma^2(t)}{\mathrm{d}t}}\mathrm{d}w \tag{11}$$

$$\mathrm{d}x = -\frac{1}{2}\beta(t)x\,\mathrm{d}t + \sqrt{\beta(t)}\mathrm{d}w \tag{12}$$

Der Rückwärtsprozess zum Generieren braucht die Score-Funktion und nimmt theoretisch die Form von Gleichung 13 an [3]. Evaluiert wird diesmal von 1 bis 0.

$$\mathrm{d}x = \left[f(x,t) - g(x)^2\nabla_x \log p(x)\right]\mathrm{d}t + g(t)\mathrm{d}w \tag{13}$$

[58] schlägt außerdem vor, Schritte der Standard Langevin-Dynamik aus Gleichung 5 in das numerische Lösen des Rückwärtsprozesses einzubauen, was die Ergebnisqualität deutlich verbessert.

4.2.1 Ein Diffusionsprozess für Graphen

Entrauschendes generatives Modellieren eignet sich besonders gut für Moleküle, weil so die Hauptherausforderung, das Diskretsein, überwunden wird. Deswegen kann das resultierende Modell vielseitig für eine Reihe von weiteren Problemen eingesetzt werden. Score-Modelle wurden schon erfolgreich für inverse Probleme [35] und für die Imitation des Cocktailparty-Effekts [33] eingesetzt. In dieser Arbeit wird später das Bearbeiten und Optimieren wichtig.
Während des Vorwärtsprozesses über einem Graphen müssen Aussagen über die Existenz von Kanten immer ungenauer werden. Die Einträge von Nullen und Einsen in der Adjazenzmatrix am Anfang können somit als Wahrscheinlichkeiten interpretiert werden. Um auf dieser Interpretation weiter aufzubauen, müssen jedoch einige Veränderungen an dem Diffusionsprozess vorgenommen werden.
Zunächst muss linear statt auf einer Kugel mit dem Rauschen interpoliert werden. Darüber hinaus darf das Rauschen nur zwischen 0 und 1 liegen. Dafür wird die Sigmoid-Funktion genutzt. Verrauschte Datenpunkte werden dann nach Gleichung 14 berechnet.

$$\tilde{x}(x_0,t) = x_0 + t\cdot\left(\frac{1}{1+e^{-z}} - x_0\right) \quad \text{mit } z \sim \mathcal{N}(0,\mathrm{I}) \tag{14}$$

Da das Rauschen durch die Sigmoid-Funktion nicht mehr normalverteilt ist, gelten die Regeln des Wiener-Prozesses nicht und eine SDGL kann nicht mehr explizit definiert und umgekehrt werden. Deshalb wird als Generationsalgorithmus die getemperte Langevin-Dynamik innerhalb der S-Funktion vorgeschlagen: So wird die Summe von normalverteilten Daten respektiert und die Tatsache genutzt, dass das Modell versucht $-z$ innerhalb der S-Funktion vorauszusagen.
Um eine Intuition aufzubauen, hilft es dennoch, die Prozesse ohne S-Funktion zu betrachten:

$$\mathrm{d}x = \frac{x}{t-1}\mathrm{d}t + \frac{1}{1-t}\mathrm{d}\bar{w} \tag{15}$$

$$\mathrm{d}x = \left[\frac{x}{t-1} - \frac{\nabla_x \log p(x)}{(t-1)^2}\right]\mathrm{d}t + \frac{1}{1-t}\mathrm{d}\bar{w} \tag{16}$$

Der Hauptunterschied zwischen der vorgeschlagenen Langevin-Dynamik und dem obigen Rückwärtsprozess ist der Divergenzterm $x/(1-t)$, der Produkt des Interpolierens ist. Da die echten Daten an den Asymptoten der S-Funktion liegen, entspricht ein großer Divergenzterm einem großen t. Betrachtet man die Verteilung der Daten mit gleichverteiltem t, so wird diese ähnlich zum VE-Prozess mit steigendem Divergenzterm immer schärfer, wenn man die Daten, wie das Netzwerk, durch die S-Funktion betrachtet.
Das Netzwerk wird unabhängig von t parametrisiert, sodass während des Generierens Schritte entgegen dem Divergenzschritt, also dem Erhöhen von t, möglich sind. Dadurch kann wie in [34] auf eine innere Schleife verzichtet werden und eine Fehlanpassung zwischen x und t ausgeschlossen werden.
Wie lang der Generationsalgorithmus braucht, um zu einem Ergebnis zu konvergieren, wird durch den Hyperparameter *eta* vor dem Divergenzterm gewichtet. Algorithmus 1 adressiert außerdem das Problem, dass die Approximation der Score-Funktion Rauschen während des Generierens hinzufügt, indem dem Wiener-Prozess durch ein geringeres λ weniger Gewicht gegeben wird. Ein $\lambda > 1$ bedeutet zusätzlich mehr Diversität, während

$\lambda < 1$ mehr Qualität ergibt, da dadurch das Gradientenverfahren in Richtung höherer Wahrscheinlichkeit wichtiger wäre.
Weitere Vorteile dieser Methode sind, dass sie im Gegensatz zu [58] nicht auf numerisches Lösen von Differenzialgleichungen und Korrekturschritte angewiesen ist und die Asymptoten bei t=1 umgeht.

Algorithm 1 Getemperte Langevin-Dynamik in der Sigmoid-Funktion

Require: $s_\theta, \alpha_0, \tau, N, \lambda$ ▷ Festsetzen der einzelnen Hyperparameter

$x \sim \mathcal{N}(0, \mathrm{I})$ ▷ Der Anfangswert stammt aus einer Standard-Normalverteilung und befindet sich außerhalb der Sigmoidfunktion.

for $n \leftarrow 1$ to N **do** ▷ Der folgende Schritt soll N mal wiederholt werden. Die Läufervariable ist n.

  $\alpha_n \leftarrow \epsilon \cdot e^{-\frac{n\tau}{N}}$ ▷ Berechnen der exponentiell abnehmenden Schrittgröße

  $z \sim \mathcal{N}(0, \mathrm{I})$ ▷ Die Richtung des Diffusionsschrittes wird einer Standard-Normalverteilung entnommen. Das Rauschen für den Adjazenztensor ist symmetrisch.

  $\nabla \leftarrow s_\theta(\mathrm{sigmoid}(x))$ ▷ Die Richtung zu höherer Wahrscheinlichkeit wird berechnet, indem x mit der Sigmoidfunktion zu den Daten aus dem entsprechenden Diffusionsprozess umgewandelt wird.

  $x \leftarrow x + \alpha_n \lambda^{-1} \nabla + \sqrt{2\alpha_n}\lambda z + \alpha_n \eta x$ ▷ x wird gemäß der Langevin-Dynamik mit Divergenzschritt aktualisiert.

return runden(sigmoid(x)) ▷ Das Ergebnis wird gerundet, sodass es zu den anderen Daten der Graphen passt.

4.3 Die Architektur

Die Architektur des KNNs ist ein ebenso wichtiger Teil wie die konkrete Methode zum Generieren, mit dem Vorteil, dass die Architektur hier auf den Diffusionsprozess abgestimmt werden kann. Die eigene Architektur ist maßgeblich von der Transformer-Architektur und dem Prinzip von neuronalen Netzen für Graphen inspiriert, weshalb diese zunächst vorgestellt werden.

4.3.1 Transformer und Aufmerksamkeit

Die Transformer-Architektur wurde ursprünglich für das Modellieren von natürlicher Sprache vorgestellt [61], wurde aber auch mit Erfolg im Bereich von Proteinsequenzen [62], Punktwolken [22] und dem computerbasierten Sehen eingesetzt [20].
Das Herz des Transformers bildet die Aufmerksamkeitsoperation. Mit ihr können die in Vektoren umgewandelten Worte gezielt Informationen austauschen. Dafür werden aus allen Worten durch gelernte affine Transformationen (auch dichte Schichten genannt) Ziel-, Angabe- und Wertvektoren Q, K und V der Dimensionalität d_K errechnet (Gleichung 17).

$$Q = W_Q x + b_Q,\ K = W_K x + b_K,\ V = W_V x + b_V \tag{17}$$

Der Zielvektor beschreibt die gewünschte Information, der Angabevektor die eigene und der Wertvektor die Nachricht. Anhand der durch das Skalarprodukt errechneten Ähnlichkeit der Angabe und Zielvektoren wird dann eine durch die Softmax und Dimensionalität normalisierte lineare Kombination der Nachricht zwischen den Wortvektoren ausgetauscht. Die genannten Vektoren werden für die folgende Gleichung entlang der Wort-Achse zu Matrizes aneinandergehängt.

$$y = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right) V \quad \text{mit} \quad \mathrm{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{d} e^{x_j}} \tag{18}$$

Die Aufmerksamkeitsoperation ist permutationsequivariant, d. h. wird die Reihenfolge der Eingabe verändert, so verändert sich die Ausgabe gleichermaßen. Um beim Modellieren von Sätzen auch die Information über die Reihenfolge beachten zu können, werden die Worte mit ihrer Position eingebettet.
Transformer benutzen wie viele vorherige Architekturen darüber hinaus Residuen, mehrere Kanäle bzw. Köpfe, Normalisierungen und mehrlagige Perzeptronen [61, 25].
Ein Problem der Transformer-Architektur ist, dass Skalarprodukte zwischen allen Worten berechnet werden müssen, was zu einer quadratischen Platzkomplexität über die Anzahl an Worten führt. Für die führenden Transformer ist es aber dennoch möglich, Sequenzen der Länge 2048 [11] zu bearbeiten. Approximationen können die Komplexität signifikant verringern [15, 46].

4.3.2 Neuronale Netzwerke für Graphen (GNNs)

In neuronalen Netzwerken für Graphen findet der gelernte Informationsaustausch zwischen den Ecken eines Graphen entlang der Kanten statt. Der Austausch ist in der Regel durch mehrlagige Perzeptronen parametrisiert und die permutationsinvariante Kombination der eingehenden Nachrichten ist eine normalisierte Summe [24]. Damit auch weiter voneinander entfernte Ecken Informationen austauschen können, wird eine Schicht ungefähr bis zum Erreichen eines Fixpunktes, den es nach dem Fixpunktsatz von Banach geben muss, evaluiert [24]. Das mehrfache Evaluieren erschwert das Fließen der Gradienten beim Training, weshalb auf Methoden der rekurrenten neuronalen Netze zurückgegriffen wird [45].
Im Gegensatz zu Transformern [61] oder faltenden neuronalen Netzen [25] schadet das Hinzufügen von weiteren Schichten schnell der Leistung. Darüber hinaus kann es passieren, dass Ecken mit wenig Abstand zu sehr ähnlichen Ergebnissen kommen (sog. „Over-Smoothing"), was ein offenes Problem darstellt [13].

4.3.3 Die MolGrad-Architektur

Die MolGrad-Architektur verbindet den durch Aufmerksamkeit geleiteten Informationsaustausch des Transformers mit der Konditionierung von Nachrichten auf die Existenz von Kanten aus den GNNs. Bemerkenswert ist, dass Kanten eigene Eigenschaften haben und im Diffusionsprozess nicht klar ist, ob sie existieren. Da das auch für vorher nicht existente Kanten gilt, werden viele der Graphen auch besonders dicht verbunden sein. Im Folgenden werden einzelne Unterschichten vorgestellt, die für genau diese Aufgabe entworfen wurden. Sie bilden den MolGrad-Block, aus welchem das Modell aufgebaut ist. Moleküle werden als eine Reihe von „one-hot"-kodierten Vektoren für die Atome und einem Adjazenztensor für die Bindungen, dessen Einträge ebenso „one-hot"-kodiert sind, dargestellt.

Vorwärts-Perzeptron

Das Vorwärts-Perzeptron (FFN) besteht aus zwei dichten Schichten der Form $y = Wx + b$ mit einer nichtlinearen Funktion wie die ReLU-Funktion ($\text{relu}(x)_i = \max(0, x_i)$) [1] dazwischen. W und b sind lernbare Parameter. In Transformern hat das Zwischenprodukt oft eine höhere Dimensionalität [61]. Der Einfachheit halber wird hier darauf verzichtet. Im Kontext von MolGrad verarbeitet ein FFN die Atomvektoren separat und ein anderes FFN jede Bindung einzeln.

Kantengesteuerte Aufmerksamkeit

Ein Zwischenprodukt des Aufmerksamkeitsmechanismus ist die Aufmerksamkeitsmatrix $QK^T/\sqrt{d_K}$. Ihre Einträge bestimmen, wie stark die Worte bzw. jetzt Atome miteinander Informationen austauschen, genauso wie die Adjazenzmatrix bestimmt, welche Atome miteinander verbunden sind.
Die kantengesteuerte Aufmerksamkeitsschicht nutzt diese Verbindung zwischen Adjazenzmatrix und Aufmerksamkeitsmatrix, um durch die Kantenvektoren informierte Kommunikation zwischen den Atomen zu lernen. Der Adjazenztensor wird zuerst durch eine dichte Schicht $\mathrm{f}(\cdot)$ auf die richtige Dimensionalität projiziert und dann zu der Aufmerksamkeitsmatrix addiert.

$$y = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}} + \mathrm{f}(A)\right) V \tag{19}$$

Die Addition in Gleichung 19 findet nach der Normalisierung und vor der Softmax-Operation statt, um die Eigenschaften der ursprünglichen Aufmerksamkeitsoperation, wie die konvexe Kombination von Wertvekto-

ren und Unabhängigkeit von der Dimensionalität der Eingabe und Varianz der Ausgabe, beizubehalten.
Vorteil dieser Schicht ist, dass dichte Verbindungen der Eingabe durch koordinierte Kommunikation für das Netzwerk nicht unübersichtlich werden. Außerdem generalisiert diese Schicht globale und lokale Kommunikation aus z. B. [71], indem sie die Wege der Kommunikation von der Eingabe abhängig macht.
Ähnliche Ideen gibt es in [73], wo der Winkel, der Abstand und die Existenz von Verbindung mit Vorfaktoren zu der Aufmerksamkeitsmatrix addiert werden. Außerdem existiert eine Transformer Architektur mit dem Namen „Realformer", die parallel zu den Worten Informationen über die Aufmerksamkeitsmatrizes, ähnlich wie die Kanteninformation in MolGrad, propagiert [26].

Neuverbinden der Kanten

Auch die einzelnen Kanten sollten auf die Informationen des restlichen Graphen zugreifen können. Separat zwischen den Kanten Nachrichten zu berechnen, würde einen erheblichen zusätzlichen Rechenaufwand bedeuten. Deswegen nutzt die vorgeschlagene Methode die Ecken: Ähnlich zum Aufmerksamkeitsmechanismus werden aus den Ecken Zielvektoren berechnet, welche gewünschte Verbindungen angeben. Dieser Vektor übernimmt auch die Aufgabe des Angabevektors, da Moleküle ungerichtete Graphen sind und somit die Aufmerksamkeitsmatrix symmetrisch sein muss. Die resultierende Aufmerksamkeitsmatix A ist die neue Adjazenzmatrix:

$$A = \frac{EE^T}{\sqrt{d_E}} \quad \text{mit} \quad E = W_E x + b_E \tag{20}$$

Extra Ecke

Es kann sein, dass wegen des Rauschen Verbindungen so getrennt werden, dass der Graph aus mehreren, nicht verbundenen Teilgraphen besteht. Zwar kann das FFN bei den Kanten dafür sorgen, dass es ein Grundniveau an Kommunikation zwischen nicht verbundenen Ecken gibt, aber empirisch zeigt sich, dass der globalen Kommunikation im Graphen noch weitergeholfen werden kann.
Ähnlich wie in [44], wird in der ersten Schicht dazu eine extra Ecke, die mit allen anderen Ecken verbunden ist, zu jedem Graphen hinzugefügt. Der Unterschied ist aber, dass diese extra Ecke nicht dem Auslesen von Voraussagen, sondern der internen Kommunikation helfen soll.
Die Verbindung zur extra Ecke hat eine zusätzliche Verbindungsart, d. h. es gibt einen extra Eintrag im „one-hot"-kodierten Verbindungsvektor. Die zusätzliche Ecke wird durch einen gelernten Vektor beschrieben.

4.3.4 Der MolGrad-Block, Köpfe, Residuen und Normalisierung

In den Aufmerksamkeits- und Neuverbindungsschichten werden wie im Transformer [61] die errechneten Vektoren Q, K und V in Vektoren geringerer Dimensionalität aufgespalten, sodass mehrere Vektoren den Aufmerksamkeitsmechanismus parallel durchlaufen können. Die Anzahl paralleler Aufmerksamkeitsberechnungen wird Köpfe genannt.
Die restliche Struktur des MolGrad-Blocks wird von dem Transformer übernommen und kann in Abbildung 2 betrachtet werden. Die Residuen ermöglichen Architekturen mit mehr Schichten und helfen der Normalisierung dabei, dass während des Trainings nützlichere Gradienten entstehen [25, 69]. Die Regularisierung durch Dropoutschichten erschwert einfaches Auswendiglernen der Trainingsdaten und verbessert somit die Generalisierungsfähigkeit. Wichtige Hyperparameter des Modells sind die Dimensionalität der Zwischenergebnisse, die Anzahl der Köpfe und die der Blöcke.

5 Experimente

In diesem Abschnitt soll MolGrad ausprobiert werden. Erst in Form eines Wirksamkeitsnachweises auf eine einfache synthetische Verteilung und dann auf Moleküle. Beim Generieren von Molekülen sollen Diversität und Plausibilität unter die Lupe genommen werden. Außerdem soll die Möglichkeit zum Bearbeiten und Optimieren bespielhaft überprüft werden.

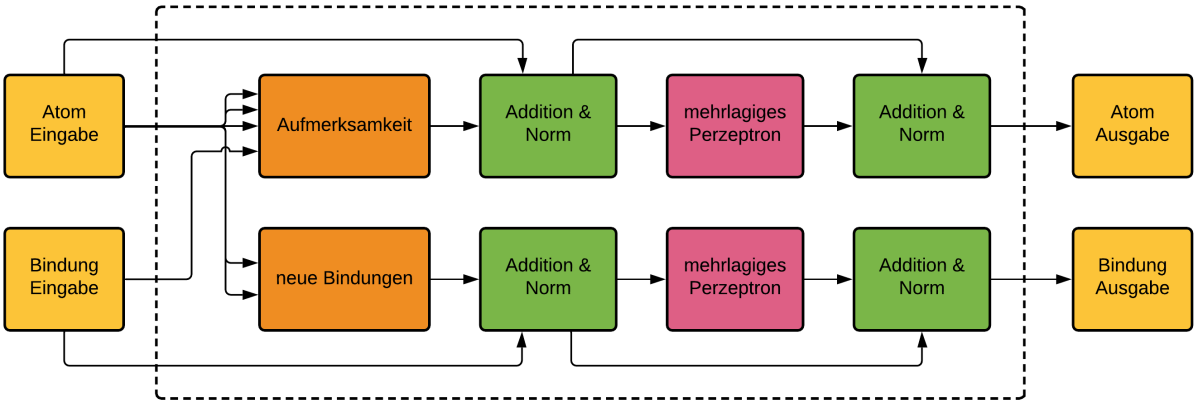

Abbildung 2: Der Berechnungsgraph des MolGrad-Blocks: Die obere Bahn manipuliert die Atome und die untere die Bindungen.

5.1 Synthetische Daten

Als einfacher Wirkungsnachweis für den neuen Diffusionsprozess wird ein zweidimensionales Beispiel gewählt. Die Einträge der Daten sind wie bei den Graphen nur Nullen und Einsen, sie werden gleichverteilt aus der Datenmenge $\mathbb{D}$ entnommen.

$$\mathbb{D} = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\} \tag{21}$$

Als Fehlerfunktion wird die Manhattan-Version der Gleichung 9 mit Formel 14 zum Berechnen der verrauschten Datenpunkte genutzt. Während des Trainings werden die Zeitpunkte t aus einer Gleichverteilung von 0 bis 1 entnommen. Der Diffusionsprozess über die synthetischen Daten ist in Abbildung 3 zu sehen. Ein Beispiel der Daten während des Trainings befindet sich im letzten Feld.
Das Modell ist ein einfaches, mehrlagiges Perzeptron mit vier Schichten aus jeweils 512 Neuronen und „leaky ReLU"[4] als Aktivierungsfunktion und unabhängig von t. Es wird für 500 Schritte mit dem Adam Gradientenverfahren [37] und einer Batchgröße von 1536 trainiert. Die Lernrate startet mit 10^{-3} und halbiert sich alle 100 Schritte. Anhand des Schritt-Fehlerfunktion-Diagramms ließ sich erkennen, dass das Modell schon nach etwa 100 Schritten in einem Minimum gelandet ist.

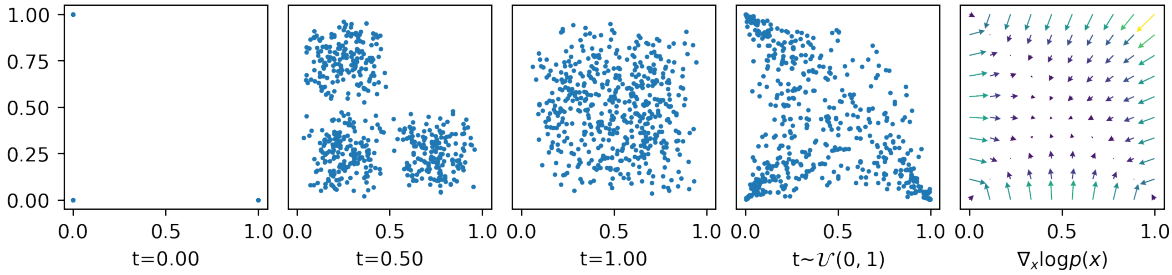

Abbildung 3: Die synthetischen Daten im Diffusionsprozess: von links nach rechts mit steigendem t. Das vorletzte Feld zeigt die Überlagerung der einzelnen Verteilungen mit gleichverteiltem t, das letzte die Score-Funktion. Die Pfeile wurden für bessere eine visuelle Klarheit nichtlinear skaliert.

In Abbildung 4, dem Generationsprozess, ist zu erkennen, dass die meiste Bewegung in den ersten Schritten passiert. Dies liegt an der exponentiell sinkenden Schrittgröße und den kleiner werdenden Gradienten. Damit die Punkte zum Schluss besser auf die Ecken verteilt sind, wird $\lambda = 1{,}05$ gewählt.

[4] definiert als $f(x) = \max(-\alpha x, x)$, hier mit $\alpha = 0.2$ [68]

Die generierten Punkte liegen alle nah an den Datenpunkten und es ist kein Punkt entstanden, der zu $(1,1)^T$ gerundet werden würde. Die Daten divergieren also nicht in alle Richtungen. Dieses Experiment weist somit das Wirkungskonzept von Algorithmus 1 in Kombination mit Gleichung 9 als Fehlerfunktion nach.

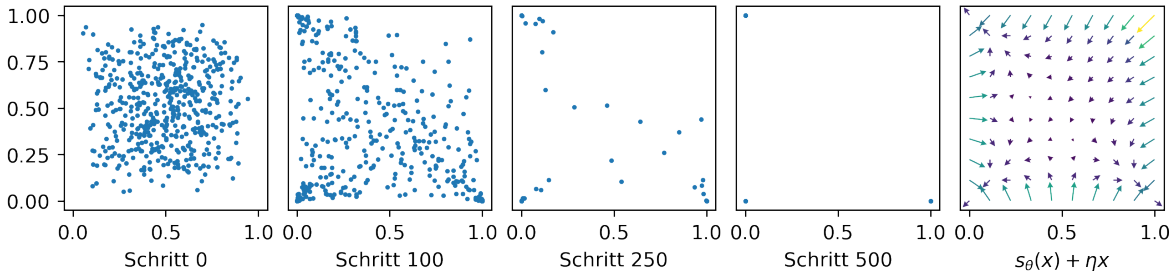

Abbildung 4: Beim Generationsprozess werden aus den zufälligen Punkten am Anfang Punkte, die den Proben aus Datensatz $\mathbb{D}$ ähneln. Im letzten Feld ist das Vektorfeld für die Generation zu sehen. Folgt man den Pfeilen wie bei einem Anfangswertproblem, so landet man bei den echten Datenpunkten.

5.2 Moleküle

Die Moleküle für die folgenden Experimente stammen aus der GDB-13 Datenbank [8], einem Datensatz aus rund 70 Millionen wirkstoffähnlichen Molekülen aus bis zu 13 Atomen der Atomsorten: C, N, O, S und Cl. Gewählt wurde diese Datenbank, da sie besonders groß und einfach verfügbar ist.

5.2.1 Generieren von 6-Atom-großen Molekülen

Für das Modellieren von größeren Molekülen wird nicht nur mehr Speicherplatz benötigt, da die Moleküle größer sind, sondern auch weil die Aufgabe schwieriger ist und somit ein Netzwerk mit mehr Kapazität benötigt wird. Um auf meiner NVIDIA GTX 1060 ein Netzwerk adäquater Größe trainieren zu können, wurden erste Experimente daher mit 6-Atom-großen Molekülen durchgeführt.
Nach einigen Versuchen mit unterschiedlichen Hyperparametern zeigt sich, dass größere Netzwerke generell bessere Ergebnisse erzielen, unabhängig davon, ob die Architektur tief oder weit ist. Es wird daher eine weite Architektur aus zwölf Blöcken mit zwölf Köpfen und einer Dimensionalität von 768 gewählt, weil so der GPU besser genutzt wird.
Das Training findet mit der AdaBelief-Methode [72] mit $\epsilon = 10^{-16}$ für besonders hohe Adaptivität beim Verfolgen der Gradienten statt. Die Lernrate wird, wie bei anderen Transformern [30], erst linear von 0 bis $8 \cdot 10^{-4}$ für die ersten 3000 Schritte aufgewärmt und sinkt dann bis zum Ende bei 60000 Schritten linear zu 10^{-5}. Die Batchgröße beträgt 64.
Der Datensatz aus den 6-Atom-großen Molekülen der GDB-13 Datenbank wird mit dem Verhältnis 9:1 in einen Trainings- und Testsatz aufgeteilt. In Abbildung 5 ist zu sehen, dass der Fehler von beiden Teilsätzen gleichermaßen sinkt, obwohl nur für den Trainingssatz optimiert wird. Das ist ein Indikator für eine gute Generalisierungsfähigkeit und bedeutet, dass das Modell nicht nur die Daten aus dem Trainingssatz auswendig lernt.

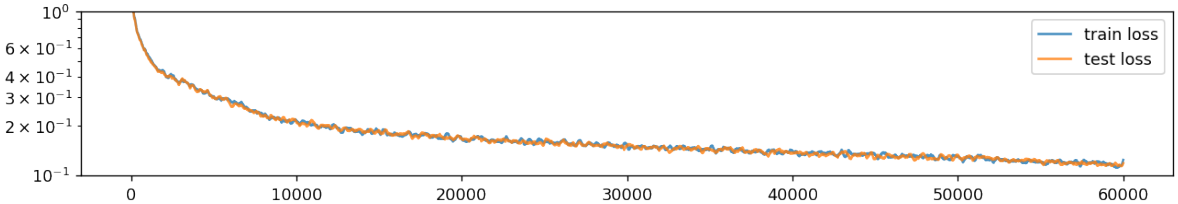

Abbildung 5: Die Ergebnisse der Fehlerfunktion (normalisiert durch die Daten-Dimensionalität) über die Trainingszeit von ca. 24 Stunden für 4000 Epochen. Die Kurve wurde mit einem Savitzky-Golay-Filter (ein Punkt alle 10 Trainingsschritte, Fenstergröße von 45 Punkten und Polynom 3. Grades) geglättet. Die beste, konstante Voraussage von immer 0 würde einen Fehler von ca. 0,8 ergeben.

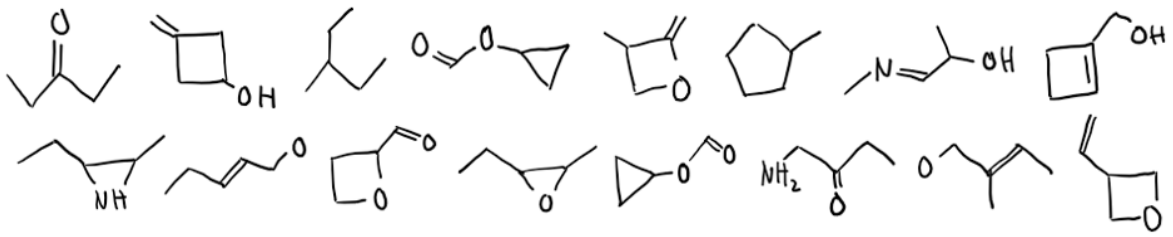

Abbildung 6: Durch MolGrad generierte, nicht handverlesene, 6-Atom-große Moleküle in der ersten Reihe und Moleküle aus dem Datensatz in Zeile 2.

In Abbildung 6 sind die mithilfe des neuen Algorithmus generierten Moleküle zu sehen. Damit ausschließlich valide Moleküle entstehen, wurde λ empirisch auf 0,5 gesetzt. Erkennbar sind dennoch eine Reihe an unterschiedlichen Molekülarten und -charakteristiken: Ketone, primäre und sekundäre Alkohole, Ester, Ether und Amine mit unterschiedlichen Ringstrukturen und Bindungen.

5.2.2 Moleküle unterschiedlicher Größe

Im Diffusionsprozess ist unklar, ob Kanten existieren. Um Moleküle unterschiedlicher Größe zu generieren wird diese Interpretation auf die Existenz von Atomen ausgeweitet. Dafür wird eine Obergrenze für die Anzahl von Atomen gewählt und kleinere Atome werden mit nicht existierenden Atomen aus Nullvektoren aufgefüllt. Für dieses Experiment werden Moleküle mit fünf bis acht Atomen aus dem GDB-Datensatz entnommen. Die restlichen Trainingsdetails bleiben bestehen.

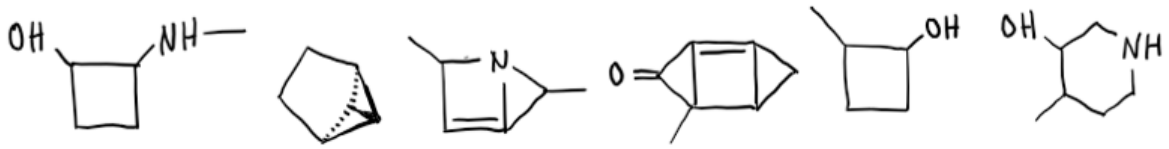

Abbildung 7: Handverlesene, mit einem Modell generierte Moleküle unterschiedlicher Größe mit $\lambda = 0{,}4$

Bei den generierten Molekülen (Abbildung 7) ist wieder eine hohe Diversität an Strukturen und funktionellen Gruppen zu erkennen und gemäß dem Datensatz bestehen die Moleküle aus unterschiedlich vielen Atomen. Das Modell scheint auch die Möglichkeit von komplizierteren dreidimensionalen Strukturen gelernt zu haben (siehe zweites Molekül von links).

5.2.3 Größere Moleküle

Auch die Fähigkeit, größere Moleküle zu entwerfen und dabei Abhängigkeiten mit großem Abstand zu modellieren, soll getestet werden. Das Modell aus den vorherigen Versuchen passt allerdings bei noch mehr Atomen nicht mehr in den verfügbaren Grafik-Speicher. Unpassend zur schwierigeren Aufgabe muss deswegen ein kleineres Netz entworfen werden. Das Netzwerk wird länger und schmaler gemacht, damit Informationen zwischen weit entfernten Atomen besser ausgetauscht werden können. Die neuen Hyperparameter sind 16 Blöcke mit acht Köpfen und einer Dimensionalität von 256.
Es wird auf Moleküle mit zehn Atomen trainiert, die restlichen Trainingsdetails bleiben gleich. Hier bedeutet das etwa 2 Epochen und ca. 35 Stunden Training. Der Fehler sinkt dennoch ähnlich zu dem 6-atomigen Beispiel.
Die generierten Moleküle in Abbildung 8 bilden unterschiedliche, bis 8-Atom-große, Ringstrukturen aus. Das Modell mit 16 Blöcken kann also wie geplant mit weitreichenden Abhängigkeiten umgehen. Die vorausgesagten Gradienten sind aber wahrscheinlich noch ungenauer, da der λ-Wert auf 0,35 verringert werden musste.

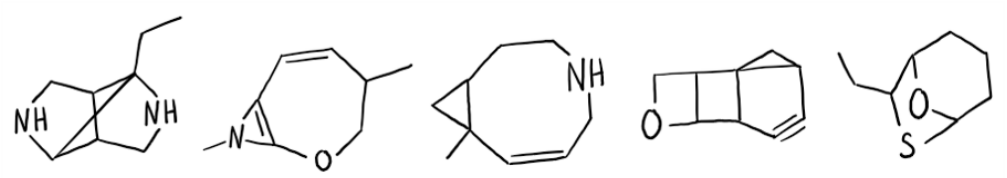

Abbildung 8: Handverlesene generierte Moleküle aus 10 Atomen. Besonders beachtenswert sind die unterschiedlichen Ringe.

5.3 Bearbeiten und Optimieren

Moleküle können durch zufällige oder gerichtete Veränderungen optimiert werden. In evolutionären Methoden werden zufällig veränderte Moleküle wiederholt evaluiert, ggf. rekombiniert und ausgewählt [41]. Gerichtetes Optimieren braucht konkrete Ideen für Verbesserungen.
Bei MolGrad kann der kontinuierliche Diffusionsprozess genutzt werden, um ein Modell $\hat{s}_\vartheta$ zu lernen, das die Ergebnisse der Evaluation im Diffusionsprozess lernt. Der Gradient zeigt dann in die Richtung von besseren Molekülen und während des Generierens kann in diese Richtung gegangen werden. Es wird also in jedem Schritt der folgende, durch λ_{opt} gewichtete Term dazu addiert:

$$+\lambda_{\text{opt}} \nabla_x \hat{s}_\vartheta(x) \tag{22}$$

Anhand von Molekül-Evaluationspaaren kann also automatisch ein Modell für gezielte Veränderungen erstellt werden. Um ein gegebenes Molekül zu bearbeiten, wird es zuerst in den Diffusionsprozess projiziert, indem das korrespondierende verrauschte Molekül zum Zeitpunkt $\hat{t}$ berechnet wird. Danach muss nur noch der Invers-Sigmoid davon genommen werden, da Algorithmus 1 innerhalb der S-Funktion stattfindet.

$$x_{\hat{t}} = \text{sigmoid}^{-1}(x_0 + \hat{t}(\text{sigmoid}(z) - x_0)) \tag{23}$$

Mit Gleichung 23 für die Initialisierung von x und Formel 22 zusätzlich in jedem Langevin-Schritt kann ein noch allgemeinerer Algorithmus definiert werden, bei dem $\hat{t}$ für die Gewichtung des gegebenen Moleküls und λ_{opt} für die Gewichtung der Eigenschaft steht. $\hat{t} = 1$ und $\lambda_{\text{opt}} = 0$ ergeben Algorithmus 1, weil beide Aspekte irrelevant wären. Eine durch den Zusatzterm erzeugte Fehlanpassung zwischen t und x ist, wie in Sektion 4.2.1 angesprochen, kein Problem.

5.3.1 Löslichkeit in Wasser

Die große GDB-Datenbank ist nicht mit Eigenschaften annotiert. Datensätze für überwachtes Lernen sind in der Regel kleiner, weil es aufwändig ist, viele Label zu generieren. Der im folgenden genutzte FreeSolv-Datensatz besteht aus ca. 650 diversen Molekülen mit experimentell ermittelten Wasserlöslichkeiten [50] und ist ein gutes Beispiel für diese Regel.
Soll ein Modell für die Eigenschaft gelernen werden ohne dabei Daten Labels nutzen, wird dies teilüberwachtes Lernen genannt. In diesem Experiment ist die Situation allerdings noch etwas komplizierter, da die Präzision des Modells für die Wasserlöslichkeit nur im Bereich des unüberwachten Datensatzes während des Diffusionsprozesses zählt.
Die Freesolv- und GDB-Datenbank betrachten sehr unterschiedliche Verteilungen an Molekülen. Im Bereich von Molekülgröße und Atomsorten unterscheiden sie sich teilweise so sehr, dass ein Transfer zwischen beiden Datensätzen unrealistisch ist. Deshalb beschränkt dieses Experiment beide Datenbanken auf Moleküle mit sechs Atomen der Sorten C und O. Das bedeutet 39 Moleküle mit Labeln und 416 ohne.
Für das Training wird die Prozedur aus [67] leicht abgewandelt: Das Lehrermodell wird auf die sauberen Molekül-Wasserlöslichkeitspaare trainiert und erzeugt dann neue Label für die GDB-Moleküle. Ein größeres Schülermodell s_ϑ lernt dann mit dem GDB-Datensatz im Diffusionsprozess, damit es während des Generierens überall akkurate Gradienten voraussagt.
Die Architektur von MolGrad wird beibehalten, nur die Voraussage wird gemäß [44] durch eine lineare Schicht aus der extra Ecke ausgelesen. Das Lehrermodell hat 10 Blöcke mit 8 Köpfen und eine Dimensionalität von

384, das Schülermodell und das generative Modell behalten die Architektur aus den ersten beiden Experimenten. Modifiziert wird das einfache Molekül Hexan mit $\hat{t} = 0{,}3$, $\lambda = 0{,}4$ und für die untere Reihe $\lambda_{\text{opt}} = 8$.

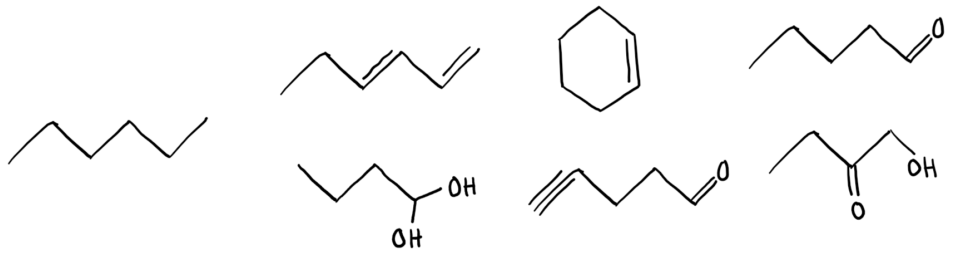

Abbildung 9: Das Ausgangsmolekül Hexan links, ohne Löslichkeitsgradienten editierte Moleküle in der oberen Zeile und optimierte Moleküle darunter; Die Wasserlöslichkeit nimmt nach rechts zu.

Um die in Abbildung 9 ersichtlichen Ergebnisse zu überprüfen wird die Log-Oktanol-Wasser-Partitionskonstante wenn möglich nachgeschlagen und ansonsten mithilfe von [4] berechnet. Wie erwartet haben die drei optimierten Moleküle mit $\{0{,}61; 0{,}54; -0{,}29\}$ deutlich geringere Ergebnisse und sind somit wasserlöslicher als alle der zufällig bearbeiteten Moleküle mit $\{3{,}16; 2{,}86; 1{,}13\}$ und der Ausgangsstoff mit 3,9. Die Ergebnisse sind nicht handverlesen.

5.3.2 Medikamentenwahrscheinlichkeit und Synthetisierbarkeit

Um in der Praxis nach für Medikamente geeigneten Molekülen zu suchen gibt, es eine Reihe an sorgfältig entworfenen Eigenschaften, wie die geschätzte Medikamentenwahrscheinlichkeit (QED) [7] und die synthetische Zugänglichkeit (SAS) [21]. Nach diesen Eigenschaften können die Moleküle in den ersten Schritten der Wirkstoffentwicklung optimiert werden, um die Anzahl der nötigen Moleküle und die aufgrund der Synthese anfallenden Kosten zu verringern. Die QED kann durch den Beitrag von einzelnen Atomen mit bestimmten Nachbarn und die SAS durch den Beitrag von einzelnen Fragmenten berechnet werden [65, 21].
Ein Netzwerk wurde dafür trainiert, beide Eigenschaften von verrauschten Molekülen vorauszusagen, sodass die Eigenschaften, während des Optimierens unterschiedlich gewichtet werden können. Das Gewicht für die QED wurde deutlich höher gewählt als für die SAS, da die Medikamentenwahrscheinlichkeit deutlich schwieriger zu optimieren schien und ansonsten von der SAS übertönt wurde ($\lambda_{\text{opt}}^{(\text{QED})} = 32$ und $\lambda_{\text{opt}}^{(\text{SAS})} = 1$). Die Ergebnisse der Optimierung sind in Abbildung 10 im Feld rechts zu sehen. Im Vergleich zu den Molekülen der Datenbank und den nicht optimierten generierten Molekülen ist ein Trend nach oben rechts zu den für die Wirkstoffentwicklung begehrenswerten Molekülen zu erkennen.

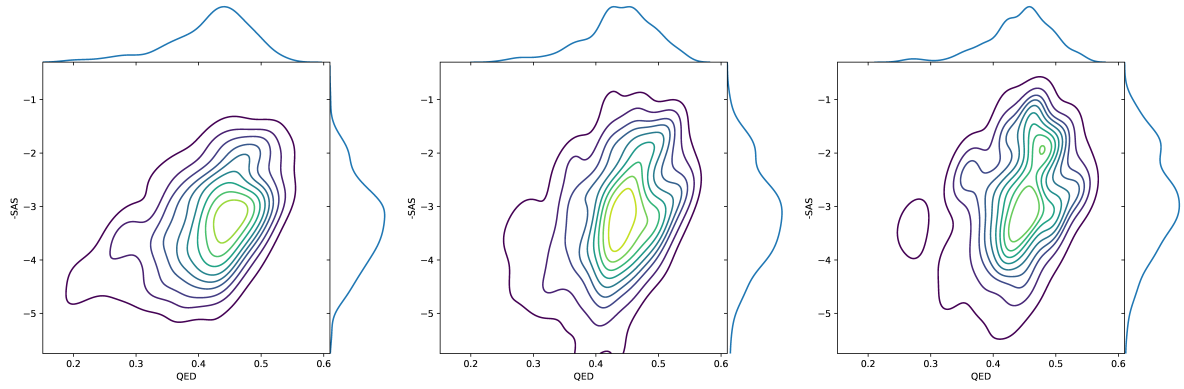

Abbildung 10: Die Verteilung der Moleküle über die Eigenschaften QED auf der x-Achse und SAS auf der y-Achse aus der GDB-6 Datenbank rechts, die generierten Moleküle in der Mitte und die nach den beiden Eigenschaften optimierten Moleküle links.

6 Fehlerfälle

Die Fehlerfälle in den von MolGrad generierten Molekülen können in drei verschiedene Klassen unterteilt werden. Der generierte Graph kann aus zwei nicht verbundenen Untergraphen bestehen, Atome können zu viele Valenzelektronen haben oder das Molekül ist von schlechter Qualität, bzw. zu unterschiedlich zu den Datenpunkten. Die ersten beiden Fälle werden mit einem Algorithmus herausgefiltert und daraufhin neu generiert. Wenn λ richtig gewählt wird, kann allerdings immer noch eine sehr diverse Menge an Molekülen generiert werden, ohne dass diese Fehler in mehr als 10% der Fälle vorkommen.
Moleküle schlechter Qualität neigen dazu, zu viele Bindungen zu haben. Das liegt daran, dass es mehr Möglichkeiten für den Wiener-Prozess gibt, Bindungen zu erzeugen als Moleküle durchschnittlich haben. Viele dieser Moleküle haben außerdem eine schlechte synthetische Zugänglichkeit und sind somit in Abbildung 10 in zweiten Feld unten zu sehen.

7 Ergebnisdiskussion und Ausblick

Im Folgenden sollen die vorgestellte Methode und die durchgeführten Experimente kritisch beurteilt werden. Wie aussagekräftig sind die Experimente? Wie kompetitiv sind die Ergebnisse und Eigenschaften?
Die Ergebnisse der Experimente haben stets die Funktionsfähigkeit des Modells demonstriert. Eine Erwartungsgemäße Einschränkung ist, dass ein Netz konstanter Kapazität größere Moleküle schlechter modellieren kann. Genauere, objektive Aussagen über die Qualität der vorgestellten Methode lassen sich aber nicht machen, weil es im Gegensatz zu Bildern [27] dafür keine weit verbreiteten, nützlichen Metriken gibt. Der Anteil an neuen, einzigartigen und validen Molekülen kann zwar ein Indiz für die Qualität sein [70], aber Ergebnisse anderer Methoden gibt es nur für Datensätze mit größeren Molekülen. Der Begrenzender Faktor für die Experimente war hier der verfügbare Grafikspeicher. Für die zukünftige Arbeit wird deshalb geplant, MolGrad am ZINC-Datensatz [32] mit professionellen GPUs zu evaluieren.
Eine andere mögliche Weiterentwicklung wäre mehr chemisches Domänenwissen in die Methode einzubeziehen. Diese Zusatzinformationen zu dem molekularen Graphen müssen allerdings entweder in dem Diffusionsprozess definiert sein, was zu einem überbestimmten Molekül als Ergebnis und somit mehr Möglichkeiten für das Netzwerk, Fehler zu machen führt. Die andere Möglichkeit wäre, dass die Architektur diese Zusatzinformationen von der Eingabe des verrauschten Graphs berechnet. Das ist allerdings selten möglich, weil Informationen wie die Bindungslängen und deren Winkel nicht für die Verrauschten Graphen definiert sind. Mit einem SE(3)-equivarianten Netzwerk mit Zugriff auf verrauschte Koordinaten könnten jedoch z. B. Informationen über die Isomerie verarbeitet werden [22].
In Sektion 3.6 wurden die unterschiedlichen Methoden darauf untersucht, wie sie zur Wirkstoffentwicklung passen. MolGrad scheint hier besser als die anderen Methoden abzuschneiden, da alle Modi abgedeckt werden und die Architektur frei wählbar und vollständig permutationsinvariant ist. Der versteckte Raum ist mit x_t implizit definiert und einfach zu interpretieren. Außerdem können Moleküle stochastisch in ihn hineinprojiziert werden. Praktisch ist auch, dass verrauschte Moleküle ohne t stehen und damit nicht überbestimmt sind. Was die eingebaute induktive Verzerrung angeht, arbeitet MolGrad direkt mit verrauschten Graphen und ist damit perfekt für leichte Veränderungen der Graph-Struktur gemacht. Der Python Quellcode ist auf `www.github.com/unitdeterminant/MolGrad` verfügbar.

8 Zusammenfassung

Inspiriert von dem kürzlichen Erfolg der Diffusionsmodelle [29, 14, 58], wurde eine neue Methode zum Generieren, Bearbeiten und Optimieren von Molekülen namens MolGrad vorgestellt. Ein Generationsalgorithmus für einen neuen interpretierbaren Diffusionsprozess wurde beschrieben und eine dazu passende Netzwerk-Architektur entwickelt, die Aufmerksamkeitsmechanismen und neuronale Netzwerke für Graphen vereint. In mehreren Experimenten wurde außerdem die Funktionsfähigkeit von MolGrad unter unterschiedlichen Bedingungen in den drei Aufgabengebieten erfolgreich nachgewiesen: Moleküle mit unterschiedlichsten funktionellen Gruppen, Ringstrukturen und verschiedenen Atomanzahlen wurden generiert und mit Techniken des teilüberwachten Lernens ist es gelungen, das Hexan-Molekül auf Wasserlöslichkeit zu optimieren. Insgesamt ist MolGrad ein funktionsfähiges generatives Modell für Moleküle mit besonderen Vorzügen im Bereich der Wirkstoffentwicklung.

Unterstützungsleistungen und Dank

Ich bedanke mich herzlich bei Angelus Dreß[5], Herrn de Vries[6] und Dagmar Wollenhaupt[7] für das Korrekturlesen.

Bilderverzeichnis

Literatur

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2019.

[2] Zaccary Alperstein, Artem Cherkasov, and Jason Tyler Rolfe. All SMILES VAE. *CoRR*, abs/1905.13343, 2019.

[3] Brian D.O. Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, May 1982.

[4] Massud A. S. Anwair, László Károlyházy, Diána Szabó, Balázs Balogh, István Kövesdi, Veronika Harmat, Judit Krenyácz, Ákos Gellért, Krisztina Takács-Novák, and Péter Mátyus. Lipophilicity of aminopyridazinone regioisomers. *Journal of Agricultural and Food Chemistry*, 51(18):5262–5270, August 2003.

[5] Richard Apodaca, Noel O'Boyle, Andrew Dalke, John van Drie, Peter Ertl, Geoff Hutchison, Craig A. James, Greg Landrum, Chris Morley, Egon Willighagen, Hans De Winter, Tim Vandermeersch, and John May. *OpenSMILES Specification*, 2007.

[6] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, 2020.

[7] G. Richard Bickerton, Gaia V. Paolini, Jérémy Besnard, Sorel Muresan, and Andrew L. Hopkins. Quantifying the chemical beauty of drugs. *Nature Chemistry*, 4(2):90–98, January 2012.

[8] Lorenz C. Blum and Jean-Louis Reymond. 970 million druglike small molecules for virtual screening in the chemical universe database GDB-13. *Journal of the American Chemical Society*, 131(25):8732–8733, July 2009.

[9] Regine S. Bohacek, Colin McMartin, and Wayne C. Guida. The art and practice of structure-based drug design: A molecular modeling perspective. *Medicinal Research Reviews*, 16(1):3–50, 1996.

[10] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. *CoRR*, abs/1809.11096, 2018.

[11] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[12] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs, 2018.

[13] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view, 2019.

[5] Mitschüler der 13. Klasse
[6] Jugend forscht Betreuer und Chemie Lehrkraft
[7] meine Mutter

[14] Nanxin Chen, Yu Zhang, Heiga Zen, Ron J. Weiss, Mohammad Norouzi, and William Chan. Wavegrad: Estimating gradients for waveform generation, 2020.

[15] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2020.

[16] Laurianne David, Amol Thakkar, Rocío Mercado, and Ola Engkvist. Molecular representations in AI-driven drug discovery: a review and practical guide. *Journal of Cheminformatics*, 12(1), September 2020.

[17] Joseph A DiMasi, Ronald W Hansen, and Henry G Grabowski. The price of innovation: new estimates of drug development costs. *Journal of Health Economics*, 22(2):151 – 185, 2003.

[18] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. *CoRR*, abs/1605.08803, 2016.

[19] Alberga Domenico, Gambacorta Nicola, Trisciuzzi Daniela, Ciriaco Fulvio, Amoroso Nicola, and Nicolotti Orazio. De novo drug design of targeted chemical libraries based on artificial intelligence and pair-based multiobjective optimization. *Journal of Chemical Information and Modeling*, 60(10):4582–4593, 2020. PMID: 32845150.

[20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

[21] Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of Cheminformatics*, 1(1), June 2009.

[22] Fabian B. Fuchs, Daniel E. Worrall, Volker Fischer, and Max Welling. Se(3)-transformers: 3d roto-translation equivariant attention networks, 2020.

[23] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[24] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, 2005.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[26] Ruining He, Anirudh Ravula, Bhargav Kanagal, and Joshua Ainslie. Realformer: Transformer likes residual attention, 2020.

[27] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[28] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *CoRR*, abs/1902.00275, 2019.

[29] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[30] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018.

[31] Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(24):695–709, 2005.

[32] John J. Irwin and Brian K. Shoichet. Zinc - a free database of commercially available compounds for virtual screening. *Journal of Chemical Information and Modeling*, 45(1):177–182, 2005. PMID: 15667143.

[33] Vivek Jayaram and John Thickstun. Source separation with deep generative priors, 2020.

[34] Alexia Jolicoeur-Martineau, Rémi Piché-Taillefer, Rémi Tachet des Combes, and Ioannis Mitliagkas. Adversarial score matching and improved sampling for image generation, 2020.

[35] Zahra Kadkhodaie and Eero P. Simoncelli. Solving linear inverse problems using the prior implicit in a denoiser, 2020.

[36] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. *CoRR*, abs/1912.04958, 2019.

[37] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[38] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions, 2018.

[39] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

[40] Geoffrey Kabue Kiriiri, Peter Mbugua Njogu, and Alex Njoroge Mwangi. Exploring different approaches to improve the success of drug discovery and development projects: a review. *Future Journal of Pharmaceutical Sciences*, 6(1), June 2020.

[41] Eric-Wubbo Lameijer, Thomas Bäck, Joost N. Kok, and AD P. Ijzerman. Evolutionary algorithms in drug design. *Natural Computing*, 4(3):177–243, September 2005.

[42] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, January 1993.

[43] Junde Li, Rasit Topaloglu, and Swaroop Ghosh. Quantum generative models for small molecule drug discovery, 2021.

[44] Junying Li, Deng Cai, and Xiaofei He. Learning graph-level representation for drug discovery, 2017.

[45] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks, 2017.

[46] Valerii Likhosherstov, Krzysztof Choromanski, Jared Davis, Xingyou Song, and Adrian Weller. Sublinear memory: How to make performers slim, 2020.

[47] Ricardo Macarron, Martyn N. Banks, Dejan Bojanic, David J. Burns, Dragan A. Cirovic, Tina Garyantes, Darren V. S. Green, Robert P. Hertzberg, William P. Janzen, Jeff W. Paslay, Ulrich Schopfer, and G. Sitta Sittampalam. Impact of high-throughput screening in biomedical research. *Nature Reviews Drug Discovery*, 10(3):188–195, March 2011.

[48] Soma Mandal, Mee'nal Moudgil, and Sanat K Mandal. Rational drug design. *European journal of pharmacology*, 625(1-3):90—100, December 2009.

[49] Lars M. Mescheder. On the convergence properties of GAN training. *CoRR*, abs/1801.04406, 2018.

[50] David L. Mobley and J. Peter Guthrie. FreeSolv: a database of experimental and calculated hydration free energies, with input files. *Journal of Computer-Aided Molecular Design*, 28(7):711–720, June 2014.

[51] Bo Pang, Tian Han, and Ying Nian Wu. Learning latent space energy-based prior model for molecule generation, 2020.

[52] Oleksii Prykhodko, Simon Viet Johansson, Panagiotis-Christos Kotsias, Josep Arús-Pous, Esben Jannik Bjerrum, Ola Engkvist, and Hongming Chen. A de novo molecular generation method using latent vector based generative adversarial network. *Journal of Cheminformatics*, 11(1), December 2019.

[53] Chongli Qin, Yan Wu, Jost Tobias Springenberg, Andrew Brock, Jeff Donahue, Timothy P. Lillicrap, and Pushmeet Kohli. Training generative adversarial networks by solving ordinary differential equations, 2020.

[54] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[55] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[56] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *CoRR*, abs/1907.05600, 2019.

[57] Yang Song, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. Sliced score matching: A scalable approach to density and score estimation. *CoRR*, abs/1905.07088, 2019.

[58] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2020.

[59] Hoang Thanh-Tung, Truyen Tran, and Svetha Venkatesh. On catastrophic forgetting and mode collapse in generative adversarial networks. *CoRR*, abs/1807.04015, 2018.

[60] Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. *CoRR*, abs/1606.05328, 2016.

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[62] Jesse Vig, Ali Madani, Lav R. Varshney, Caiming Xiong, Richard Socher, and Nazneen Fatema Rajani. Bertology meets biology: Interpreting attention in protein language models, 2020.

[63] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, July 2011.

[64] Max Welling and Yee Teh. Bayesian learning via stochastic gradient langevin dynamics. pages 681–688, 01 2011.

[65] Scott A. Wildman and Gordon M. Crippen. Prediction of physicochemical parameters by atomic contributions. *Journal of Chemical Information and Computer Sciences*, 39(5):868–873, August 1999.

[66] Olivier J. Wouters, Martin McKee, and Jeroen Luyten. Estimated Research and Development Investment Needed to Bring a New Medicine to Market, 2009-2018. *JAMA*, 323(9):844–853, 03 2020.

[67] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V. Le. Self-training with noisy student improves imagenet classification, 2020.

[68] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network, 2015.

[69] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization, 2019.

[70] Chengxi Zang and Fei Wang. Moflow: An invertible flow model for generating molecular graphs. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Jul 2020.

[71] Shuo Zhang, Yang Liu, and Lei Xie. Molecular mechanics-driven graph neural network with multiplex graph for molecular structures, 2020.

[72] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James S. Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients, 2020.

[73] Łukasz Maziarka, Tomasz Danel, Sławomir Mucha, Krzysztof Rataj, Jacek Tabor, and Stanisław Jastrzębski. Molecule attention transformer, 2020.