Skoltech

Skolkovo Institute of Science and Technology

MASTER'S THESIS

Optimized molecular grammars for in silico drug discovery

Master's Educational Program: Center for Computational and Data-Intensive Science and Engineering

Student

Mikhail Frolov
Center for Computational and Data-Intensive Science and Engineering
June 19, 2020

Research Advisor:

Maxim V. Fedorov
Full Professor

Moscow 2020

Skoltech

Skolkovo Institute of Science and Technology

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Методы минимизации молекулярных грамматик для вычислительного поиска лекарств

Магистерская образовательная программа: Центр по научным и инженерным вычислительным технологиям для задач с большими массивами данных

Студент_____

Михаил Фролов

Центр по научным и инженерным вычислительным технологиям для задач с большими массивами данных

Июнь 19, 2020

Научный руководитель:_____

Максим В. Фёдоров

Профессор

Москва 2020

Optimized molecular grammars for in silico drug discovery

Mikhail Frolov

Submitted to the Skolkovo Institute of Science and Technology
on June 19, 2020

Abstract

Designing molecules with desired properties is the primary goal of molecular optimization. Due to hard chemical constraints, generating molecules with desired properties is not trivial. Generated structures may not be valid or may not satisfy specified constraints. Thus, the task of generating molecules is very difficult.

There are many ways to represent the molecular structure in a computer science. One of the most common is the SMILES representation - a string representation of a molecular graph. Using this representation, researchers have developed many methods for generating molecules using linear graph notation.

Obviously, the task of generating a molecular structure is solved by combining the variational autoencoder approach and the Bayesian optimization of latent space. This approach has technical and fundamental limitations: the generation of one incorrect symbol can make the whole structure incorrect; many structures are generated without chemical meaning (sequences of carbons). Unfortunately, this method does not generate 100% valid molecules. There are a number of methods for generating molecular structures using graph notations, but they require more computational resources and have very complicated programming part.

The developed representation of LegoGram molecular grammars could not only to disassemble the molecule into its constituent parts, but also to assemble it from these parts, and the chemical validity of the assembled molecule is guaranteed by the grammar itself. In addition,obtained grammar could be optimized by the average length of the representation of molecular structures and the number of molecules themselves. Improved grammars allow to compress the representation of molecules in half compared to conventional SMILES. It was also shown that the use of molecular grammars allows recurrent models of neural networks to achieve state-of-the-art solutions with 100% molecular validity. Optimized molecular grammars can further reduce the representation of the molecule, as well as represent the chemical meaning in the generation of structures.

Research Advisor:
Name: Maxim V. Fedorov
Degree: Doctor
Title: Full Professor

Методы минимизации молекулярных грамматик для вычислительного поиска лекарств

Михаил Фролов

Представлено в Сколковский институт науки и технологий
Июнь 19, 2020

Реферат

Моделирование молекул с заданными свойствами является основной задачей молекулярной оптимизаци. Из-за жестких химических органичений задача генерирования молекул с заданными свойствами не является тривиальной. Сгенерированные структуры могут быть невалидны или не удовлятворять заданным ограничениям. Таким образом задача генерации молекул является весьма непростой.

Существует множество способов представить молекулярную структуру в компьютере. Одной из самых распространенных является представление SMILES - строковое представление молекулярного графа. Используя это представление исследователи разработали множество методов генерации молекул используя линейную нотацию графа.

Отчасти, задача генерации молекулярной структуры решается с помощью совмещения вариационого подхода и байесовской оптимизации латентого пространства. Данный подход имеет технические и фундаментальные недостатки: генерация одного неверного символа может сделать всю структуру неверной, генерируется множество структур без химического смысла (последоавельность углеродов). К сожалению, такой метод не дает генерации 100%. Существует ряд методов генерации молекулярных структур при помощи графовых натаций, однако их использование сопряжено со значительными затрататами вычислительных ресурсов, а также трудоемко с точки зрения программирования.

Разработанное представление молекулярных грамматик LegoGram позволяет не только разобрать молекулу на составные части, но и собрать её из этих частей, причем химическая валидность собранной молекулы гарантируется самой грамматикой. К тому же, данные грамматики можно оптимизировать по средней длине представления молекулярных структур и количеству самих молекул. Получшенные грамматики позволяют сжать представление молекул вдвое по сравнению с обычными SMILES. Также было показано, что применение молекулярных грамматик позволяет рекуррентным моделям нейросетей достигать state-of-the-art решений со 100% валидностью молекул. Оптимизированные молекулярные грамматики позволяют ещё уменьшить представление молекулы, а также предстаавляют химический смысл при генерации структур.

Научный руководитель:
Имя: Максим В. Фёдоров
Ученое звание, степень: Доктор наук
Должность: Профессор

Acknowledgments

Firstly, I am thankful to Sergey Sosnin for consultations and explanation of basic theory of chemoinformatics. Also I am grateful to my scientific supervisor Maxim Fedorov for discussions and explanation this theory from a different angle. I respect other members of Molecular Science Group and Skoltech Center for Computational and Data-Intensive Science and Engineering for invaluable knowledge and useful instructions.

Secondly, I am grateful for Ivan Khokhlov for developing approach to construct molecular grammars. His comments are very useful. Also I want to express my gratitude to Zhores supercomputer team for provision of computing resources.

Contents

List of Figures

Chapter 1

Introduction

Rapidly development of deep learning area had an impact to all spheres of our live. Unmanned vehicle [Shakhatreh et al., 2019, Grigorescu, 2019], chatbot robots [Mnasri, 2019, Kar and Haldar, 2016], smart technology are some of the few examples of this. Of course, deep learning had a very significant impact to chemistry [Jr, 2019, Goh et al., 2014]. Part of chemistry, which studies information technology methods in chemistry is chemoinformatics.

In another hand, producing new candidates for cutting the edge drug and material design area is very expensive and time-consuming. So, this area needs experimentally and computation methods for searching for molecules. One could view this problem as an optimization problem, in which an algorithm should search the molecules with desired properties. But this task is extremely challenging due to heterogeneity and complexity of chemical space. Nowadays only $\approx 10^8$ chemical structures have been synthesized and potential drug-like candidates are estimated between 10^{23} and 10^{60}. That's why the brute-force approach is not feasible here.

Virtual screening is the chemoinformatics method that is used for the selection of potential drug candidates. One of the important tasks in chemoinformatics is the Quantitative structure-activity relationship problem(QSAR). It is a fundamental property prediction problem of chemical compounds. There is another, the maybe more important problem of generation of compounds with desired properties(inverse-QSAR/QSPR).

In chemoinformatics, a molecule can be represented as a molecular graph. There many approaches to representing the molecular graph in the computer. One popular methods is to use the linear notation of molecular graphs - simplified molecular-input line-entry system (SMILES). SMILES is an ASCII string, which specifies chemical structures. This notation was initially designed to balance between representation accuracy and human readability

When molecules represented as SMILES, then one could pass the ball to another machine learning branch - natural language processing. Natural language processing is a part of machine learning, which works with documents on natural language, for example - English. Natural language processing is a well-studied area of machine learning, but it is significantly progressed in the deep learning era. One of the first approaches of deep learning was the implementation of a recurrent neural networks, such as LSTM. This approach was good for generating simple SMILES string, but there were issues with generating big molecules, and producing molecules with desired

properties is very complicated.

So, generating molecules with desired properties is an important task for virtual screening. Gomez-Bombarelli et al. have conducted breakthrough research in this area [Gómez-Bombarelli et al., 2018]. The team has demonstrated a variational autoencoder approach in chemistry and shows to the community the way for optimizing chemical space to generate molecules with desired properties. Unfortunately, this method not without flaws. Molecules generation is more complex tasks than generate sentences. For example, if LSTM produces one incorrect word in the text it can be insignificantly for human. But if neural network produces one incorrect symbol in SMILES string, a molecule will be invalid and there is no simple way to fix it. Another problem with this approach is less chemical sense in the generation of molecules. A typical example of such a molecule is a sequence of carbon atoms. Chemically it is a valid molecule but very simple.

Thus variational autoencoder approach generates drug-like molecules, but it fails many times. The number of correctly generated molecules is low. Kusner et al. have considered syntactic validness of generated SMILES strings and have concluded that one could build grammar to produce syntactic correct SMILES strings [Kusner et al., 2017]. This approach was implemented with variational autoencoder and have improved previous results by a significant margin. Nevertheless, this approach hasn't solved the syntactic correctness of generated molecules.

The evolution of graph neural networks also has impacted chemoinformatics. Firstly graph neural networks were used for solving the forward QSAR problem. This type of neural could conduct a molecule not as a linear sequence but as some object in the non-euclidean manifold. GNN approach allows using of this space to consider molecules from the different sides and extract of useful features.

Generating molecules using graph neural networks is an extremely challenging task. Jin was the first who reported 100% validity of generating molecules by building junction tree variational autoencoder [Jin et al., 2018]. He has used graph message passing neural network to achieve this result. So, Jin et al. have solved the decoder error issue, but the semantic validity of molecules wasn't been solved.

Later Kajino et al. proposed molecular hypergraph grammar to solving this issue [Kajino, 2019]. He considers each molecule graph as a molecular hypergraph and builds a parsing tree according to molecular hypergraph grammar. This approach also has solved the decoding error issue, moreover, it produces semantic valid molecules.

Kajino et al. approach are very good and for now, it is a state-of-the-art method to produce molecules with desired properties. But it is very theoretically difficult and one will have some issues with implementing such an approach. So, we propose another application of graph grammars - node-label controlled grammar. This grammar is much simpler for understanding and we have proposed the framework for embedding this grammar to neural networks. We consider molecules

as lists of grammar rules which could be combined together to construct a molecule.

1.1 Motivation

The scientific community and companies need new methods for artificial search for drugs. Existing text string processing methods are bad because chemical compounds are more structured than text. Recurrent neural networks do not allow the implementation of this structural knowledge in its construction.

Therefore, we decided to develop a new method for the structural construction of molecules. Despite the fact that there are a large number of methods, they are difficult not only to understand but also computationally.

Our motivation was to develop a good method for generating 100% semantically valid molecules. Another problem is the redundancy of the SMILES representation. SMILES generates at the character level, not at the level of functional groups. This presentation can be compressed by highlighting the most common blocks.

Chapter 2

Literature review

2.1 Neural networks

Unfortunately, people's health isn't so good. Many people take medicines and of course, drug design is a multi-billion business [Dubois and Mouzon, 2014]. Actually, many drug-like compounds are rejected after clinical expertise. This is very bad for business and for the community because companies are wasting money and people are losing time. So, generating molecules that are actually drug-like, well- synthesized, and have some adjusted properties is a very important problem for our world [Bielska et al., 2011, Lionta et al., 2014].

On the other hand, we are living in a new technology era [Mélanie-Bécquet et al., 2015]. Computers, Internet, Spaceships and other high-technological device has been built by people from the second part of XX century [Leong et al., 2014]. Also significant goals are achieved by neural networks [Raghu and Schmidt, 2020, Ozbayoglu et al., 2020, Li et al., 2020]. Starting from perceptron invention, they up to the new level after winning in Kaggle competition of object detection. Neural networks has become meaningful parts of our life. We use neural networks every day. They are in our phones [Lee et al., 2019, Wang et al., 2020], predicting good things for us, make our pictures more beautiful and so on.

There are many studies that are connected with ANNs [Buhrmester et al., 2019]. Hebb, McCulloch, and Pitts performed first studies about ANNs [Cowan, 1989]. There are was the first ideas of implementing the neurons model in silico. Unfortunately, first experiments with these types of models haven't shown good performance and were rejected for many years. Later, in 1982, Hopfield described activation functions [Hopfield, 1982] and then Werbos introduced back-propagation algorithm [Werbos, 1990]. Now the back-propagation algorithm is the most popular algorithm for training neural network. Without exaggeration, one could say that many state-of-the-art neural networks are propagated via back-propagation. This is a very simple-understandable algorithm and this fact impact its popularity. In chemistry one of the first applications of ANNs was applying a multi-layer perceptron to study chemical engineering processes by Hoskins [HOSKINS, 1991]. After MLP was used to predict the secondary structure of proteins.

One could imagine an artificial neural network as a mathematical technique, which is based on human brain functioning. All ANN share the concept of neurons. Neurons consist of dendrites,

axons. Model of perceptron [Rosenblatt, 1958] is pretty like to real biology neuron. Each neuron in an artificial neural network has its own analog in biology - ”synapse”. Each human neuron consists of dendrites, cell body, and axons [Zhang, 2019]. This biological architecture has a pretty mathematical interpretation. Dendrites are entries, the cell body is calculating the weighted sum of these entries, and the axon is analog of calculating some non-linear functions.

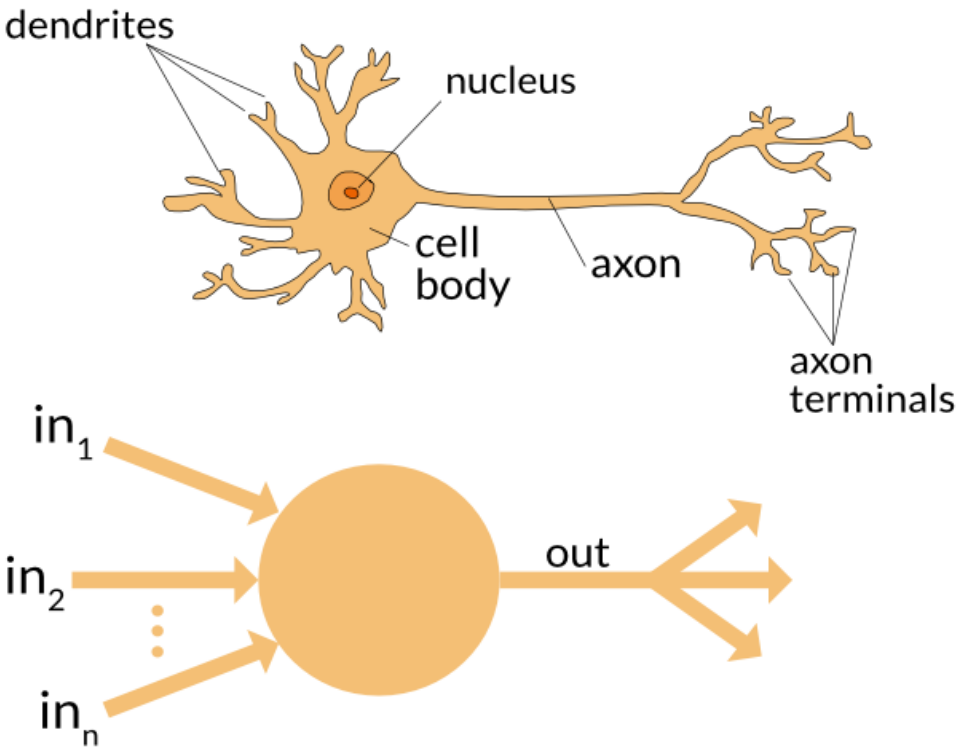

Figure 2.1: Neuron in biology

Increasing applications of artificial neural networks were the main force in developing new architectures. Nowadays many architectures are successfully used by people. Yann LeCun in 1998 developed a very good model for recognizing handwritten digits [Lecun et al., 1998]. Also, there is another type of ANN - recurrent network.

2.2 Recurrent neural networks

Unfortunately, a simple perceptron isn’t efficient in sequence to sequence tasks. For these types of problem special neural network architecture was investigated. There’s some kind of magic of this type of neural network. Let’s go deeper.

One could say, that problems such as a sequence in input or sequence in output are rare and don’t need to be attended, but if one has built NN in vector manner (all inputs and outputs are vectors), that’s it could be processed in a sequential manner and seq-to-seq formalism could be used.

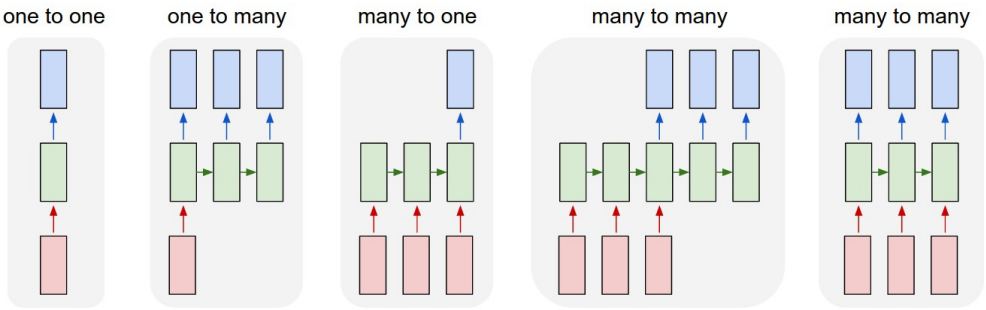

Figure 2.2: Different types of recurrent networks

2.2.1 RNN computation

In contradiction to simply fully connected neural networks, where each layer has one matrix and one bias vector - one layer of simply RNN has three matrices.

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y = W_{hy}h$$

The typical scheme of using recurrent neural networks is presented in the picture. One has to prepare encoding for our sequence. One approach is one hot encoding (on the picture there is one hot encoding) [Potdar et al., 2017], this approach is good for the low dimension of the dictionary, but for a big dictionary, this approach isn't good. The first layer of the trainable matrix will be very big and probably it couldn't fit into the memory.

Another option of making this embedding - train matrix that fit our dictionary not in dummy one hot embedding but in the more compressed representation [Singh et al., 2019]. And our network produces numbers that could be associated with probabilities of outputting symbols of sequence.

2.2.2 LSTM and GRU

Vanilla RNN is a good to approach for solving time series problems or making seq-to-seq modeling. But actually we training RNN with backpropagation algorithm [Lecun, 2001]. One should understand that backpropagation algorithm is a gradient-based algorithm. There are two issues with gradient-based and these gradients could vanish or explode during time [Lipton et al., 2015, Tang and Glass, 2018]. If the gradients explode, the model couldn't be trained. If they vanished, long-term sequences couldn't be learned because gradients after n'th step will be zero or another small value. Let's consider this problem more closer. Let s_t be state vector and Δv - change of vector v caused by change in vector s. We want to obtain conditions for:

$$\lim_{k->\infty} \frac{\Delta s_{t+k}}{\Delta s_t} = 0.$$

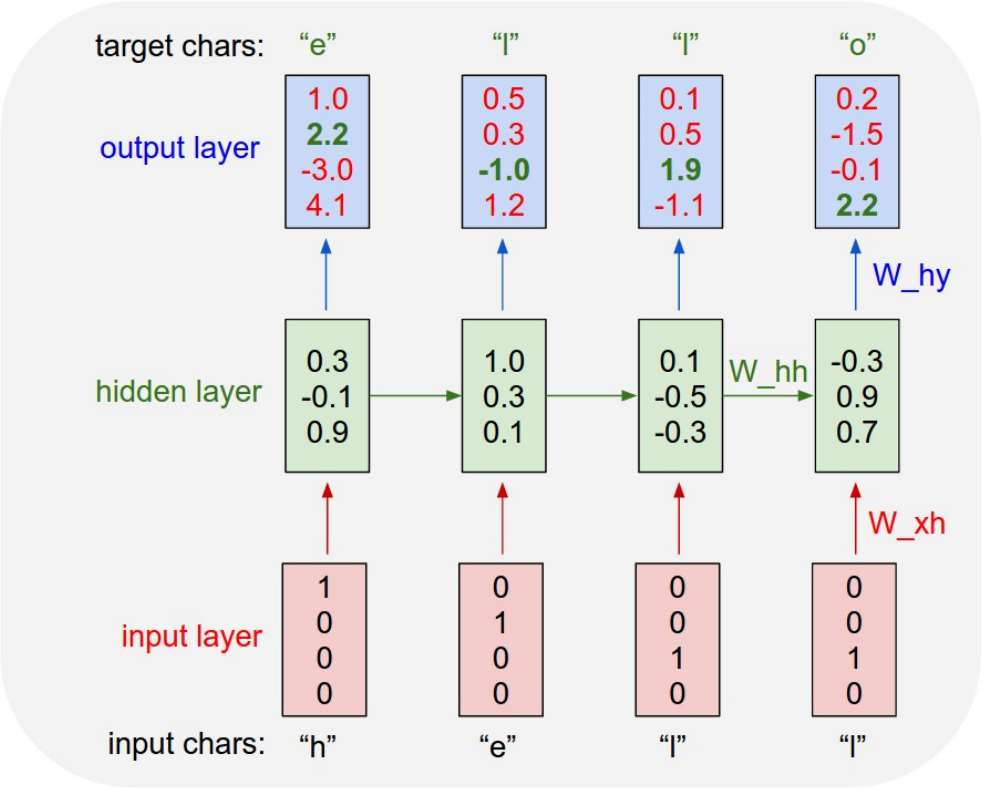

Figure 2.3: Example of RNN

Pascanu proved that the same sufficient conditions could be obtained by [Pascanu et al., 2012]:

$$\lim_{k->\infty} \frac{\partial s_{t+k}}{s_t} = 0.$$

Vanilla RNN could be formalized by $s_{t+1} = \phi(z_t)$ where $z_t = Ws_t + Ux_{t+1} + b$. Apply mean value theorem for limit $c \in [z_t, z_t + \Delta z_t]$:

$$\Delta s_{t+1} = [\phi'(c)]\Delta z_t = [\phi'(c)]\Delta(Ws_t) = [\phi' c]W\Delta s_t.$$

Declare that

$$\gamma = \sup_{c \in [z_t, z_t + \delta z_t]} \|[\phi\|'(c)]\|.$$

For logistic sigmoid $\gamma \leq \frac{1}{4}$. For euclidean norm of vector Δs_{t+1} on each step one obtain:

$$\|\Delta s_{t+1}\| = \|[\phi'(c)W\Delta s_t\| \leq \|[\phi'(c]\|\|W\|\|\Delta s_t \leq \gamma\|W\|\|\Delta s_t\| = \|\gamma W\|\|\Delta s_t\|.$$

And $\Delta s_{t+k} \leq \|\gamma W\|^k \|\Delta s_t\|$:

$$\frac{\Delta s_{t+k}}{\Delta s_t} \leq \|\gamma W\|^k.$$

If $\|\gamma W\| < 1$, then $\frac{\Delta s_{t+k}}{\Delta s_t}$ decreases exponentially. If $\|\gamma W\| < 1$ when $\|W\| < 4$ for sigmoid case and for tanh case $\|W\| < 1$. In vanilla RNN there is only one approach: we can initialize weights.

Let's find a good initialization for weights:

$$\|\gamma W = 1\|$$

Consider tanh case, so: $\gamma = 1$, then:

$$\|W\| = 1$$

By definition of operator norm:

$$\|W\| = 1$$

iff $|We_i| = |e_i| = 1$. Then:

$$\sum_i w_{ij}^2 = 1$$

Let's generate w from random distribution in $[-R, R]$ and there are n entries in column j. So:

$$n\mathbb{E}(w^2) = 1$$

$$\mathbb{E}(w^2) = \frac{R^2}{3}$$

as variance of random variable w. Then we get:

$$n\frac{R^2}{3} = 1$$

$$R = \frac{\sqrt{3}}{\sqrt{n}}$$

Then we have obtained good initialization weights:

$$R = [-\frac{\sqrt{3}}{\sqrt{n}}, \frac{\sqrt{3}}{\sqrt{n}}].$$

We have obtained result pretty closer to Xavier-Glorot initialization for perceptron:

$$[-\frac{\sqrt{6}}{\sqrt{m+n}}, \frac{\sqrt{6}}{\sqrt{m+n}}].$$

A vanilla recurrent neural network is a good mathematical tool for predicting time-series elements. But there is one problem, which we haven't solved in the previous step. There are two problems: exploding and vanishing gradients. This happens because we multiply equal small or big matrices. For solving this problem Hochreiter proposed LSTM - Long Short Term Memory - special kinds of recurrent neural networks [Staudemeyer and Morris, 2019, Mangal et al., 2019, Sherstinsky, 2020]. Here the scheme of basic LSTM network:

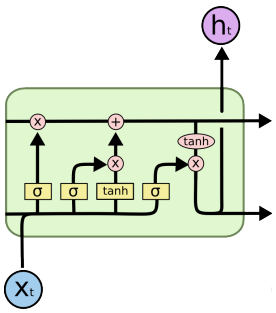

Figure 2.4: Scheme of LSTM

Basic scheme of LSTM is presented on figure 2.4. The horizontal line on this scheme is a skeleton of the idea of LSTM. The key idea of LSTM is cell state. The last cell state passes through the cell and is changed only by multiplication and summation of corresponding gates. Due to the lack of application of activation function on this line gradients in the backpropagation algorithm, less vanish [Ribeiro et al., 2019].

The first step of LSTM is to deciding what part of input should be discarded. At this step input and last hidden state are concatenated and have passed through a linear layer with sigmoid activation:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

In the next part of the cell, we decide what part of new information we want to store in the cell. This part has two sub-parts: input gate layer and a tanh layer which decides what part of our data we want to update:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$\widetilde{C}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$$

Previous state cell is updated by the rule:

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t,$$

where $*$ means point-wise multiplication. In multiplication by f_t we decide what part we reject and what new value we want to add to current cell.

The last part has reminded. We have to decide what we should output. In this part of the cell, concatenated input data are passed through a sigmoid linear layer. Then it point-wise multiplicated with the current cell state. So, at this step input data multiplicated with the current cell state which the cell decided to save.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

2.2.3 Gated Recurrent Unit

Obviously, that basic LSTM configuration aren't unique. There are many another variants of LSTM cell [Guo et al., 2019]. There are LSTM with "peephole connections" or variation of LSTM, where forget and input gates are coupled. But the most awesome variant of LSTM is Gated Recurrent Unit or GRU proposed by Cho et al(2014) [Chung et al., 2014, Che et al., 2018, Turkoglu et al., 2019, Cho et al., 2014]. This type of network also was developed to solve the vanishing gradient problem. In this variant forget and input gate is presented by one update gate. So, GRU is faster than LSTM and requires less memory. Let's consider the GRU layer. It has the update gate, reset gate, and current memory content. The update gate decides which part of the past information should pass through.

$$z_t = \sigma(W_z[h_{t-1}, x_t])$$

Obviously, the reset gate decides what part of information should be rejected from the cell.

$$r_t = \sigma(W_r[h_{t-1}, x_t])$$

Also, this variant of LSTM has its current memory content. It uses a reset gate to store important information from previous cells. Tanh function is used because we want that our values should be in the range $[-1, 1]$.

$$\widetilde{h_t} = \tanh(W[r_t * h_{t-1}, x_t]])$$

And the final part is used for output next hidden state of the network. It uses information from the update gate to determine what part of the information from current memory content and what part from previous steps should go further. The information from the reset gate used to calculation current memory content.

$$h_t = z_t * h_{t-1} + (1 - z_t) * \widetilde{h_t}$$

As we can see this type of neural network also solve the gradient vanishing problem because the new value of hidden state calculates without direct apply activation functions(only Hadamard product of two vectors).

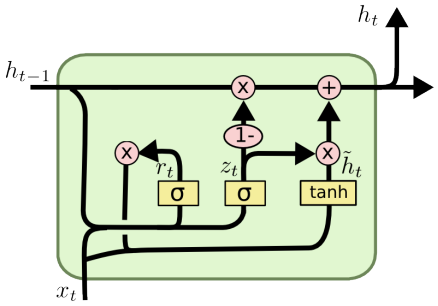

Figure 2.5: Scheme of GRU

2.2.4 Comparison between LSTM and GRU

Continue our discussion about LSTM and it's variant GRU one should provide some comparison. Obviously, if all considered cells are solving gradient vanish problems, why we should have so many variants of it.

Chung et al. was a pioneer in considering these types. In this paper, his team made experiments with vanilla RNN, LSTM, and GRU on different datasets. The task he considered was polyphonic music modeling and speech signal modeling. Data was provided by academic researchers and the Ubisoft company. He reported that on some tasks GRU is better, in some LSTM, but both LSTM and GRU outperform vanilla RNN. So, it is expected. In vanilla RNN vanishing gradient problem hasn't solved.

But what variant of LSTM researchers should use for studies? Short answer: "It depends". It depends on tasks, on data, from variance and homogeneity of data and from other things. So, researchers should have some intuition about their tasks. For this study, GRU has been used due to it's efficiency and speed of training.

2.3 Neural Network Generation architectures

Obviously, many methods of generation molecules exist [Elton et al., 2019, Mansimov et al., 2019, Pölsterl and Wachinger, 2019, Popova et al., 2019, Jin et al., 2020, You et al., 2018]. The simplest one is the random search algorithm: algorithm generates characters symbol by symbol until we obtain valid molecule [Gao and Coley, 2020]. But one could observe the important fact of molecule generation: one incorrect symbol could brake molecule and it will be invalid [Siami-Namini et al., 2019]. So this brute force is good only for the simple molecules. One of the rules for drug-like molecules could be formulated as atom count should be in the range between 20 and 70. Due to this rule the brute force method becomes inefficient for generating drug-like molecules [Lipinski et al., 2001, Lipinski, 2004, Oprea et al., 2001, Leeson and Springthorpe, 2007, Leo et al., 1971, Ghose et al., 1999, Congreve et al., 2003]. Another algorithm for generating drug-like molecules

is generating them by genetic algorithm. This is a more suitable algorithm for generating drug-like molecules, due to some manipulating with drug molecules, not creating it from scratch [Sadeghi et al., 2014, Whitley, 1994, Harik et al., 2006, Coffin and Smith, , Echegoyen et al., 2013, Sadowski et al., 2013, Taherdangkoo et al., 2013].

On the other hand, with the growing popularity of neural networks, researchers have started to apply neural networks for generating molecules [Deutsch, 2018, Lu, 2019, Gregor et al., 2015]. One of the first attempts to do this is using recurrent neural networks [Graves, 2013, Su et al., 2019, Demir et al., 2019]. As described in the early section, recurrent neural networks have to use gates or another technique to solve the gradient vanishing problems. In 250k dataset mean number of array approximately 50 in SMILES notation [Weininger, 1988, Byers et al., 1990, Neglur et al., 2005, Byers et al., 1990]. So in the average neural networks have to correct predict 50 characters without any mistakes [Chaudhari et al., 2019]. For this task, one could use LSTM or it's variations for generating molecules, but there is another problem, which we will face: researchers are interested in models, which generate molecules with desired properties or something constraints. Recurrent neural networks could generate molecules in some distribution (which we trained), another option is training RNN with initialization hidden layer of the molecule. This produces pretty good results for produce molecules, but the variation of generated molecules is huge.

Neural networks develop very fast and rapidly and researchers have created many neural network architectures for generating new structures.

2.3.1 Variational autoencoder

Generative vs discriminative modeling is one major part of deep learning practice [Ardizzone et al., 2020, Agnese et al., 2020, Giri et al., 2016, Wang and Raj, 2015]. A generative (or agent) model simulates generation data in the real process. Discriminative one has to distinguish generated sample from a real sample. Important note: in physics, chemistry, biology, and other studies modeling is almost always generative one [Wang and Raj, 2015, Sawada et al., 2019, Krusinga et al., 2019].

A variational autoencoder is a network, specially designed for creating statistically independent and semantically meaningful unsupervised representation learning task [Kingma and Welling, 2019, Doersch, 2016, Ghosh et al., 2019]. This is an auxiliary task. VAE is two connected but independently parametrized neural networks: encoder and decoder. The encoder part is the discriminator and the decoder is a generator in auxiliary terms. Encoder provides posterior data, then data is transformed into meaningful variational space. After the generative part update it's weighted inside EM - algorithm [Roche, 2011, Tolooshams et al., 2019, Zhuang et al., 2019].

One of the main advantages of probabilistic models is that they contain some uncertainty of knowledge of the desired phenomenon [Masegosa et al., 2019, Sun and Bischl, 2019, Jing and Xu,

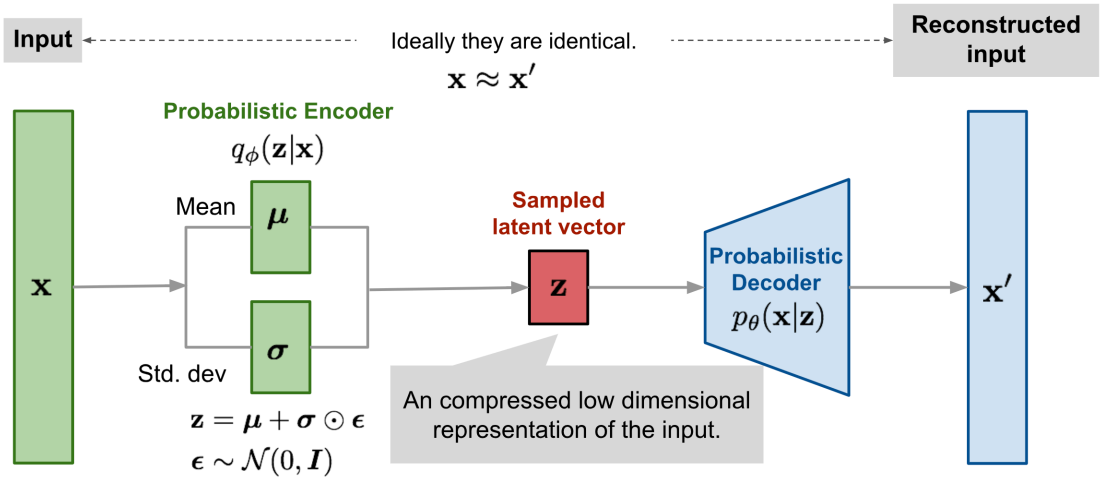

Figure 2.6: Explanation of Variational Autoencoder

2019, Sun and Bischl, 2019]. This uncertainty could be specified by a distribution which model uses. Also, there is a very important fact, that these models could contain continuous-valued distribution and discrete-valued distribution. The ideal model will specify all bias and variations of the phenomenon by specifying jointly probability distributions of these variables. The common denotation of the random sample is $\boldsymbol{x}$. This random sample is generated from the unknown underlying process. Let's denote probability of this prior distribution $p^*(\boldsymbol{x})$. Of course, it is unknown. Our task is to approximate this distribution by our model $p_{\boldsymbol{\Theta}}(\boldsymbol{x})$ with parameters Θ:

$$\boldsymbol{x} \sim p_{\boldsymbol{\Theta}}(\boldsymbol{x}).$$

By learning we mean the process of searching Θ parameters, such that $p_{\boldsymbol{\Theta}}(\boldsymbol{x})$ approximate $p^*(\boldsymbol{x})$ for any observed $\boldsymbol{x}$:

$$p_{\boldsymbol{\Theta}}(\boldsymbol{x}) \approx p^*(\boldsymbol{x})$$

.

There are many criteria for evaluating probabilistic models, but the most common is the maximum log-ikelihood [van Opheusden et al., 2020, Li and Malik, 2018, Vinayak et al., 2019, Arvinte et al., 2019]. In the training procedure, our aim is to find parameters Θ that maximize the average (or sum) of the log-probabilities of the data in the model. Log-probability is given by the equation:

$$logp_\Theta(D) = \sum_{x \in D} logp_\Theta(x),$$

where:

$$p_\Theta(x) = \prod_{j=1}^{M} p_\Theta(\boldsymbol{x_j}|P_a(x_j),$$

$$\boldsymbol{\eta} = NeuralNet(P_a(\boldsymbol{x}))$$

$$p_{\Theta}(x|P_a(\boldsymbol{x})) = p_{\boldsymbol{\Theta}}(x|\eta)x$$

This loglikelihood has minimized via stochastic gradient descent, which uses random mini-batches of a subset of our dataset. This produces an unbiased estimation of loglikelihood for the whole dataset.

The next generation of deep probability models is deep latent variable models [Kim et al., 2018, Livne et al., 2020, Ding and Gimpel, 2019, Chung et al., 2015]. They are using some variables, which hidden from user. Common denotation of latent variable is z. The result distribution of the variables x could be formulated as a joint distribution of observed and hidden variables:

$$p_{\Theta}(\boldsymbol{x}) = \int p_{\Theta}(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z}.$$

For understanding, how variational autoencoder works let's consider deep latent variable model for multivariate Bernoulli data and latent variable sampled from spherical Gaussian latent space:

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; 0, \boldsymbol{I}$$

$$\boldsymbol{p} = NeuralNet(\boldsymbol{z})$$

$$logp(\boldsymbol{x}|\boldsymbol{z}) = \sum_{j=1}^{D} logp(x_j|\boldsymbol{z}) = \sum_{j=1}^{D} x_j \log p_j + (1 - x_j) log(1 - p_j)$$

In this formulas p is a probability, so $|p| <= 1$.

In the previous paragraphs, variational autoencoder was introduced as a neural network, which consists of two parts encoder and decoder network. Encoder model is a deep latent variable model, which aim is to interpret input data to a conditional probability from latent variables:

$$q_{\phi}(\boldsymbol{z}|\boldsymbol{x}) \approx p_{\Theta}(\boldsymbol{z}|\boldsymbol{x}).$$

Training the encoder part, we should calculate such parameters ϕ, which will parametrize distribution $q_{\phi}(\boldsymbol{z}|\boldsymbol{x})$. In variational autoencoder encoder parametrize distribution as a normal distribution class:

$$(\mu, log\sigma) = Encoder_{_{phi}}(\boldsymbol{x})$$

$$q_{\phi}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, diag(\sigma)).$$

Due to the variational autoencoder have to produce examples like in the dataset, but not the same. For this problem, evidence lower bound method loss has used [Ji and Shen, 2019, Odaibo,

2019, Sobolev and Vetrov, 2019]. Let's consider $logp_\Theta(\boldsymbol{x})$:

$$logp_\Theta(\boldsymbol{x}) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[logp_{\Theta(\boldsymbol{x})}]$$

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

$$= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[log\frac{p_\Theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[log\frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p_\Theta(\boldsymbol{z}|\boldsymbol{x})}\right].$$

This loss is combined from two paths: variational lower bound and Kullback-Leibler divergence [Bulinski and Dimitrov, 2019, Nielsen, 2019, Shlens, 2014, Al-Labadi et al., 2020]. Variational lower bound make autoencoder produce the same result, KL divergence makes the distributions not close to each other. The main advantage of ELBO is the fact, which one could optimize ϕ and Θ simultaneously. It means that decoder and encoder are training together. So the pipeline of the variational autoencoder could structurized in these parts.

Unfortunately, ELBO in basic form couldn't be calculated by backpropagation rule, but reparametrized one could be calculated with backpropagation [Xu et al., 2018, Wilson et al., 2017, Tokui and sato, 2016]. A common way to do it in represents it via univariate Gaussian distribution.

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

$$(\boldsymbol{\mu}, log\sigma) = Encoder_\phi(\boldsymbol{x})$$

$$\boldsymbol{z} = \boldsymbol{\mu} \odot \epsilon$$

This repametrization let rewrite posterior density is:

$$logq_\phi(\boldsymbol{z}|\boldsymbol{x}) = \sum_i log\mathcal{N}(\epsilon_i; 0, 1) - log\sigma_i.$$

Also another one advantage of this approach is that it could be extended to the full covariance variant:

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

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{L}\boldsymbol{\epsilon},$$

L is a lower triangular matrix, with non-zero entries of the diagonal. All formulas and conclusions are the same as in the noncovariance one.

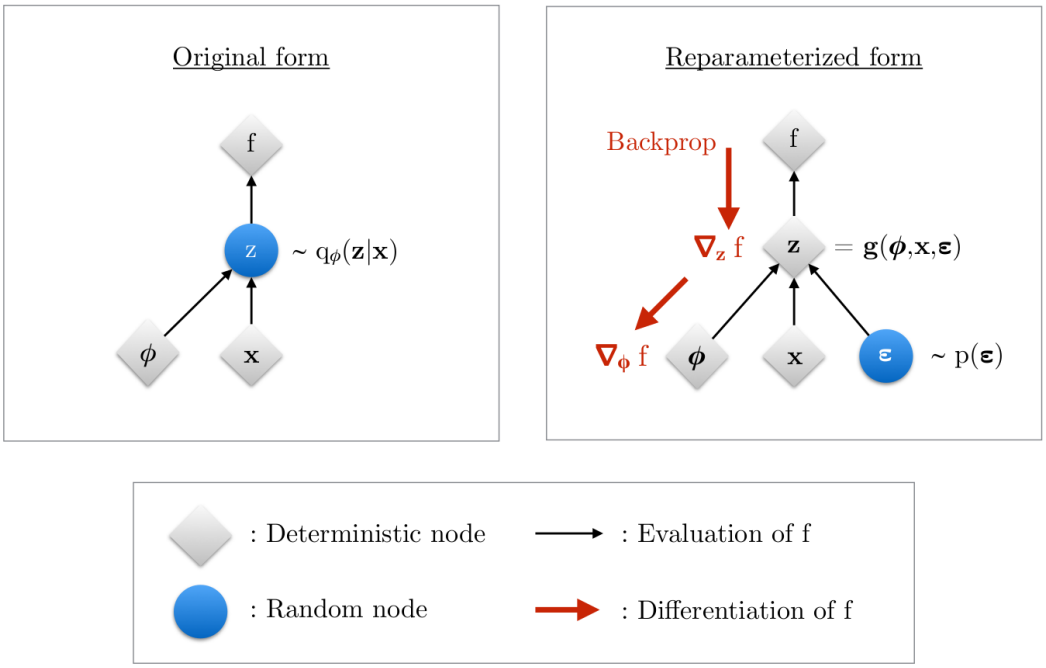

Figure 2.7: Reparametrization trick

2.3.2 Reinforcement Learning

Chemoinformatics has many ways to generate molecules. One of these way is reinforcement learning [Arulkumaran et al., 2017, Kaelbling et al., 1996, Shao et al., 2019]. As in variational autoencoder reinforcement learning approach is also adversarial. The main advantage of reinforcement learning is that it doesn't need a supervisor. So, the requirement for differentiability is not significantly here.

The aim of reinforcement learning is to optimize reward [Arora and Doshi, 2018]. The reward R_t is a scalar function that shows how the agent has approached victory or how well was the agent step t. The reward should be formulated in terms of reward hypothesis, such as all goals of the agent (and our algorithm) is formulated as a maximization problem of the reward function. As stated previously, the reward function could be not differentiable. Thus, our main goal is to maximize the reward function.

The environment in reinforcement learning is a place where agent is trained. The history H_t is a sequence of steps. And the state is the information about environment. One type of state is the Markov state:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t].$$

So, the state is a Markov state if it doesn't depend on the previous history.

The agent plays a key role in Reinforcement Learning. The agent makes steps and obtains reward. The agent has three major components:

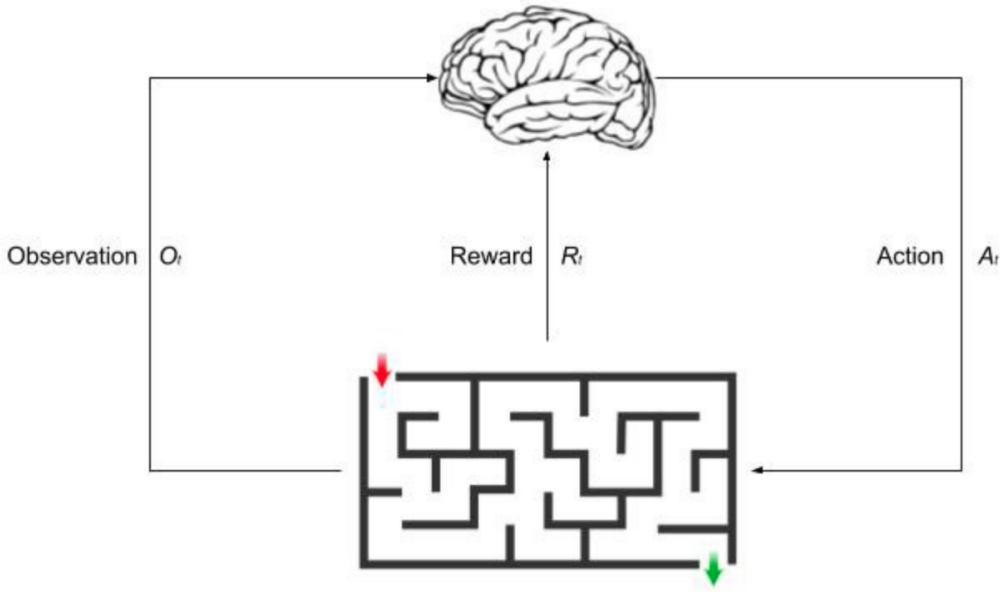

Figure 2.8: Scheme of reinforcement learning

- **Policy:** It is a function which maps current state to action. The policy could be deterministic:

$$\alpha = \pi(s),$$

  or stochastic:

$$\pi(\alpha|s) = \mathbb{P}[A_t = a|S_t = s]$$

- **Value Function:** make the prediction for future reward:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s]$$

- **Model:** knowledge about the environment. Model predicts what environment will do next:

$$R_s^a = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$$

Actually, these components are absolutely not necessary to be together. There are many types of RL agents classified by the existence of these parts: value-based, policy-based, model-based/free, and something else.

2.4 Grammar Variational Autoencoder

Generative models on SMILES is good actually, but humanity always strove to do better. The disadvantage of basic Variational autoencoder is the low number of correct molecules. According

to molecular rules, one error in SMILES string could destroy the whole molecule. Thus, generating big molecules is not trivial task for Variational Autoencoder.

Developing ideas of the basic method, Kusner improved char VAE [Kusner et al., 2017]. Each molecule represents not in SMILES format but in a parse tree from context-free grammars. A context-free grammar is tuple $G = (V, \Sigma, R, S)$, where V is a finite set of non-terminal symbols, Σ is the finite set of terminal symbols, R is a finite set of production rules and S is a start symbol. In this way, the number of validly generated molecules has rapidly increased.

The main difference between char VAE is the decode and encode procedure. To encode molecule, firstly molecule should be parsed for rules. Then, from these rules researcher construct the parse tree. This tree describes in what order we will apply rules. After this tree will be decomposed by a sequence of production (how rules were applied). Then we encode this sequence a 1-hot encoder. Obtain data we should pass to variational autoencoder.

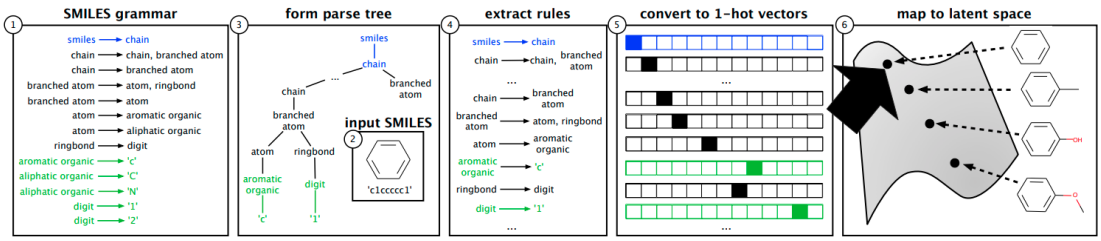

Figure 2.9: Encoder of GVAE [Kusner et al., 2017]

The decoder is simply the inversed decoder. The goal of the decoder is to produce as little as possible invalid molecules. We sample vector from latent space, then convert it to logits. The most interesting part follows next. For rules which do not satisfy our mask, we construct mask and sample with recalculated probabilities. This procedure we will do until at least one non-terminal exists. So, the main benefit of this decoder is that it produce syntactically valid SMILES.

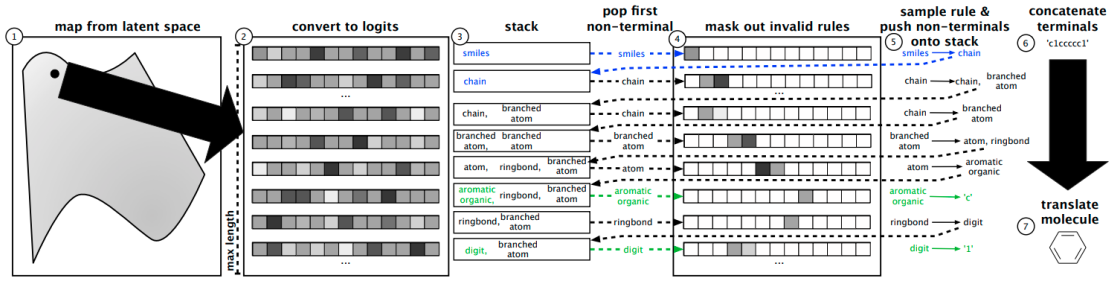

Figure 2.10: Decoder of GVAE [Kusner et al., 2017]

But this decoder has one disadvantage. It produces syntactically valid SMILES, but not semantically. Obtained molecules could have no chemical sense, could be very unstable or chemical invalid. Another disadvantage of this approach is that it doesn't control nested cycles or other nested objects. In original paper, Kusner has reported that he obtained 53.7% valid molecules on reconstruction. So, this is a good improvement but not without flaws.

2.5 Graph Neural Networks

Deep learning is a very fast-growing field of science [Ying et al., 2018, Zhou et al., 2018]. There are many types of neural networks, feed-forward neural networks, convolution networks, recurrent networks. But usually, data for these types of networks are presented in euclidean space. Nevertheless, remains a lot of tasks, which data are presented in non-euclidean space. Typically class of problems in non-euclidean is problems on graphs. For these tasks there is a new generation of neural networks: graph neural networks.

The success of convolution neural network demand to transfer the convolution operator to graphs. Graph convolutions are divided into spatial-based and spectral-based. The most simple is a spatial graph neural network. It has a direct analogy with a simple convolution on the grid, where neighbors of each node are considered. The best illustration for this is presented on the figure.

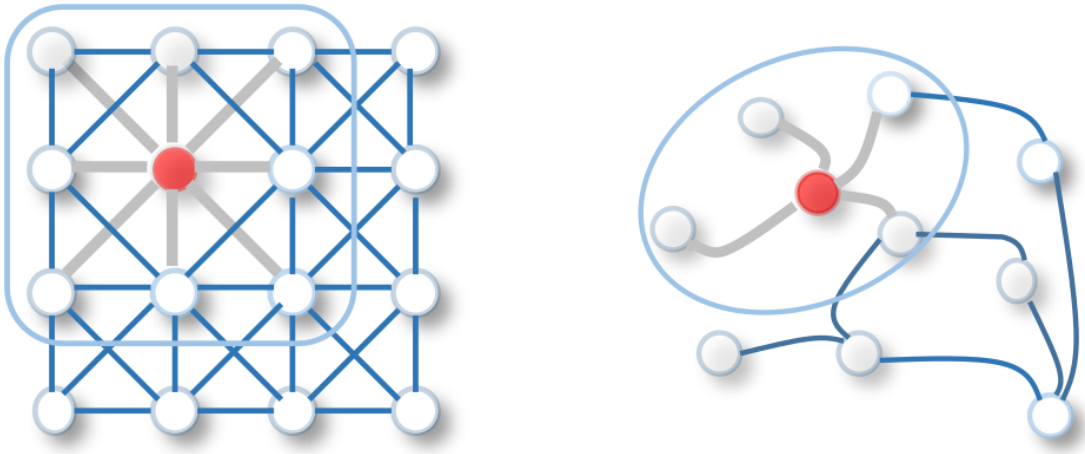

Figure 2.11: 2D Convolution vs. Graph Convolution

Another type of convolution is spectral based convolution. Omitting the derivation, one could formulate spectral graph convolution as follows:

$$H_{:,j}^{(k)} = \sigma(\sum_{i=1}^{f_{k-1}} U\Theta_{i,j}^{k} U^T H_{:,i}^{k-1}),$$

where k is the layer index, $H^{(k-1)} \in \boldsymbol{R}^{n*f_{k-1}}$, f_{k-1} is the number of input channels and f_k is the number of output channels, $\Theta_{i,j}^{(k)}$ is a diagonal matrix with learnable parameters and $U \in \boldsymbol{R}^{n\times n}$ is the matrix of eigenvectors of matrix $L = I_n - D^{-\frac{1}{2}}AD^{\frac{1}{2}}$, where A is the adjacency matrix of such graph [Wu et al., 2020]

Graph neural network shows the new state-of-the-art result for forwarding QSAR problem [Zhou et al., 2018, Pope et al., 2018, Olsen et al., 2019, Pham et al., 2018]. Due to its view from the non-euclidean manifold, a neural networks could generate new features for predicting models.

But inverse QSAR problems are more complex and approaches, which using this technique, are considered below.

2.6 Junction Tree Variational Autoencoder

All considered earlier approaches have used SMILES linear notation of molecules. This notation actually has its pros and cons. The main advantage of this notation is that it is string notation and it could be read by a human. So, researchers could use a natural language processing approach to molecules and obtain pretty good results.

The increasing the popularity of graph neural networks gave a new impetus to the VAE approach in chemoinformatics. Also, it is very important to say that no one is interesting in generating molecules, but researchers have interested in generating molecules with desired properties or satisfying some conditions. The main contribution of junction tree variational autoencoder is the direct realization of molecular graphs. For generating molecular structure variational we should construct a tree-structured scaffold and then use the VAE approach.

One could imagine that a junction tree is a structured object whose role is indicating the scaffold sub-graphs and some relation between them. One of the main problems in this method to choose the clusters of atoms, such as a set of these clusters will represent all molecules from the dataset. The main theory of this approach have based on the definition of junction tree: junction tree $\mathcal{T}_G = (V, E, X)$ is a connected labeled tree whose node-set is $V = C_1, ..., C_n$ and edge set is E. In its turn, cluster $C_i = (V_i, E_i)$ is an induced sub-graph with some constraints:

1. The union of all clusters equals source graph.

2. For all clusters C_i, C_j and C_k, $V_i \bigcap V_j \subseteq V_k$ if C_k is on the path from C_i to C_j.

Encoder and decoder is a message-passing neural networks. Each message in these nets is updated by GRU layers. After sampling from this variational encoder there is no way to obtain invalid molecule. Jin has reported that he obtained 76.7% reconstruction and 100% validity. So, this is a very good result, and first, where 100% valid molecules could be sampled. The disadvantage of this method is it's complex [Jin et al., 2018].

2.7 Molecular Hypergraph grammar VAE

Another stater-of-the-art method is the molecular hypergraph grammar model. Jin et al. have solved the decoder issue and have reported 100% validity of molecules. But his idea of clusters doesn't consider which atom in the cluster connected to another cluster. So, stereochemistry information disappears.

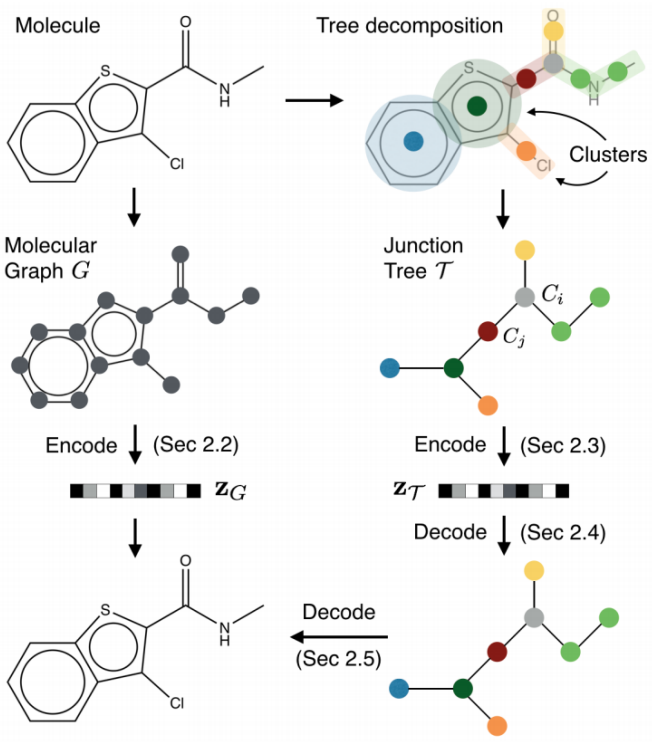

Figure 2.12: Review of Junction Tree method [Jin et al., 2018]

Kajino proposed context-free grammar which never generates invalid molecule and preserves stereochemistry information. For this purpose molecular hypergraph grammar has been used. A molecular hypergraph grammar could be defined as a hyperedge replacement grammar that always generates molecular hypergraphs. Hyperedge replacement grammar is a context-free graph grammar generating hyperedge by replacing a non-terminal hyperedge with another hypergraph. When molecular subhypergraphs will combine, they will preserve stereochemistry information and could be combined according to this information.

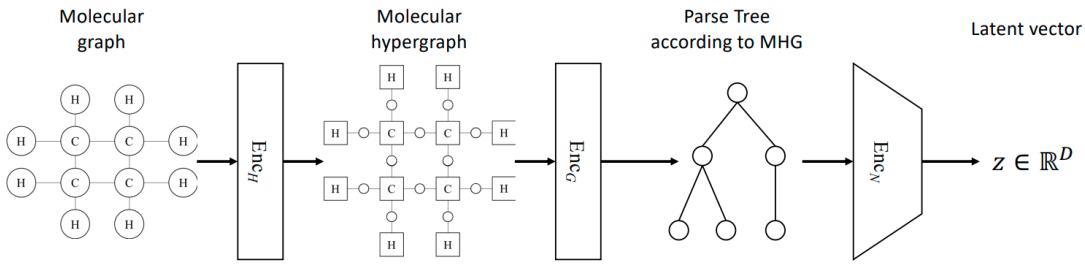

Figure 2.13: Molecular hypergraph encoder [Kajino, 2019]

As a VAE method, molecular hypergraph grammar has an encoder and decoder. Due to complexity of model both encoder and decoder consist from three-part: first encoder make a molecular hypergraph from a molecular graph, then the second encoder translate molecular hypergraph to parse tree and, finally, the third encoder produce latent vector from this parse tree.

Kajino has reported that his method produces 94.8% reconstruction error and of course

100% valid. This technique beat the previous one, but has the same disadvantage: it is too complicated.

Chapter 3

Methodology

3.1 Formal Grammars

For explaining formal grammars first go to the definitions.
Alphabet - a finite set of symbols. By a symbol, we mean any elements on which we are going to build a grammar.
Chain of symbols in alphabet V is any finite set of sequences of this alphabet. A chain which does not consist of any elements is an empty chain. The empty chain we will denote as ϵ.
If α and β - chains, then chain $\alpha\beta$ is a concatenation of chains α and β.
The reverse of chain α is a chain, which symbols are written in reverse order.
n -th order of chain α is a n concatenated chains α:

$$\alpha^n = \underbrace{\alpha\alpha...\alpha\alpha\alpha}_{n}$$

Length of chain is a number of symbols consisting in chain.
Let V^* - set consisting all chains in alphabet V including empty chain ϵ.
Let V^+ - set consisting all chains in alphabet V excluding empty chain ϵ. So, $V^* = V^+ \cup \{\epsilon\}$
Cartesian square $A \times B$ is a set $\{(a, b)|a \in A, b \in B\}$.
An ordinary formal grammar if a quadruple G = (Σ, N, R, S) in which:

- Σ is the alphabet of the language is a finite set of symbols, from which the strings in the language are built. The second name of this is a terminal symbol.
- N is a finite set of category symbols representing the syntactic categories defined in the grammar. Every symbol denote some property that every element in Σ is deemed to have or not to have. Actually, it could be any property. For example syntactic types or nonterminal symbols or something else, it depends on grammar. $N \cap \Sigma = \emptyset$
- R is a finite set of grammar rules $(\Sigma \cap N)^+ \times (\Sigma \cap N)^*$, each representing a possible structure of strings with the property A $\in$ N as a concatenation $X_1, .., X_l$ of zero or more symbols

$X_1, .., X_l \in \Sigma \cup N$, with $l \geq 0$

$$A \rightarrow X_1 ... X_l$$

- $S \in N$ is a distinguished category symbol representing all well-formed sentences defined by the grammar.

Language generated by grammar G is a set $L(G) = \{\alpha \in \Sigma^* | S \Rightarrow \alpha\}$. Grammars are equal if the generate equal languages.

3.1.1 Chomsky grammar classification

Type 0 Any generating grammar which satisfies the definition above is a type 0 grammar. Type 1 A formal grammar is a context-sensitive if all rules in R are of the form: $\alpha A \beta \rightarrow \alpha \gamma \beta$, where $A \in N, \alpha, \beta in (N \cap \Sigma)^* and \gamma \in (N \cap \Sigma)^+$ Type 2 A formal grammar is a context- free, if every rule from P could be written as:
$A \rightarrow \beta$, where $\beta in (T \in N)^*$ Type 3 A right regular grammar is a formal grammar G, such that all the production rules in R are of one of the following forms:

1. $A \rightarrow a$, where $A \in N, a \in \Sigma$
2. $A \rightarrow aB$, where A and B are non-terminals in N and a is in Σ
3. $A \rightarrow \epsilon$, where A is in N and ϵ denotes the empty string.

A left regular grammar is a formal grammar G, such that all the production rules in R are of one of the following forms:

1. $A \rightarrow a$, where $A \in N, a \in \Sigma$
2. $A \rightarrow Ba$, where A and B are non-terminals in N and a is in Σ
3. $A \rightarrow \epsilon$, where A is in N and ϵ denotes the empty string.

A regular grammar(type 3) is a left or right regular grammar. Definition of the parsing tree

Let $G = (\Sigma, N, R, S)$ be a grammar and consider trees of the following form. Each node of the tree is labeled with a symbol form $U \cup N$, and its sons are linearly ordered. A node labeled with $a \in \Sigma$ must have no sons. For each node labelled with $A \in N$, let $X_1, .., X_l$ be the labels in its sons; the the grammar must contain a rule $A \rightarrow X_1, ..., X_l$. The yield of a tree is a string $w \in \Sigma^*$ formed by all nodes labeled with symbols in Σ, written according to the linear order. If X is the label of the root, such a tree is called a parse tree of w from X.

3.2 Molecular Grammars

A graph is a pair $G = (V, E)$, where V is a set whose elements are called vertices, and E is a set of vertices, whose elements are called edges. In math molecules considered molecular graphs. The molecular graph is connected undirected graph one-to-one corresponded to the structural formula of the chemical compound so that vertices of the graph correspond to atoms of the molecule and edges of the graph correspond to chemical bonds between these atoms. A chemical graph is a labeled graph whose vertices correspond to the atoms of the compound and edges correspond to chemical bonds. Its vertices are labeled with the kinds of the corresponding atoms and edges are labeled with the types of bonds. Chemists often the consider hydrogen-depleted molecular graphs. The hydrogen-depleted molecular graph is the molecular graph with hydrogen vertices deleted (and corresponding edges). Molecular graphs can not distinguish between geometrical isomers, but can successfully distinguish structural isomers.

Of course, there is a big interest in scientific society and technology companies to formalize and produce some rules (grammars) to generating graphs indirect way. The formal specification of graph grammars is based on graph rewriting or graph transformation. A graph rewriting system consists of a set of graph rewrite rules $L \rightarrow R$. L here is called pattern graph and R being called replacement graph. The graph rewriting step corresponding to insert commands can directly be taken from grammar rules. Another operation which has to be provided by graph grammar is delete operation. Any graph has to have the starting (or anchor) node. Graph delete operation has to be applied in increment manner, which means that all graph nodes have to be deleting before delete root node. This is also true for sub-graphs in the current graph.

Graph grammars are often divided into two groups: node replacement grammar and hyperedge replacement grammar. In node replacement grammars they are consist of rules that define how a single node can be replaced by a sub-graph. In hyperedge replacement grammars the rules are showing how a hyperedge can be replaced by a graph.

A rewrite rule in a node replacement grammar is of the form $N \rightarrow \frac{G}{E}$, where N is a node label, G is a graph, E is an embedding rule. Applying the rule to a graph G consists of taking a node x with label N, removing all its incident edges, replacing the node with S, and reconnecting S to the nodes originally connected to N according to the rules specified in E. In the case of NLC grammars, E is a set of couples (a, b) where a and b are node labels.

Hyperedge replacement grammar is another type of graph grammar. A hyperedge is an atomic item with a fixed number of tentacles, called the type of the hyperedge. It can be attached to any kind of structure comes with a set of nodes by attaching each of its tentacles to a node. The hyperedge controls the sequence of these attachment nodes and can play the role of a place holder, which may be replaced with some other structure eventually. When hyperedge replacement occurs,

the hyperedge is removed and the replacing structure is attached to the current node.

3.3 LegoGram

Hyperedge replacement grammar applied to molecular science had been done by Kajino. It was the next stage of producing molecules via the Bayesian technique. This team has produced the next state-of-the-art result and successfully solved syntactic and partly semantic issues.

This work is about another branch of graph grammar rules - node replacement grammar. We have considered node-label controlled graph grammar. It is the simplest case of graph grammars. In node-label controlled graph grammar the replacement process is local to the desired node. Embedding rules of this grammar describe only the mechanism of how to connect S graph to the neighborhood of the non-terminal N. Node-labeled controlled grammar could be written as a tuple $G = (\Sigma, \Delta)$, where Δ is an alphabet of node-labeled sub-graphs and Δ is a rule how this labeled nodes could be connected.

We have implemented node-labeled controlled grammar in the molecular science area. Its name is LegoGram. It is much simpler then molecular hypergraph grammar. Also, it can be built in any RNN-based neural network. Also, some features are implemented such as mutator or 100% generator of molecules. The main parts of the algorithm are encoded and decode. Encode function is here:

Algorithm 1: Encode algorithm

```
Result: Array of numbers, which represent rules
Input  : Molecular graph
Output: Embedded rules
initialization;
obtain rings info;
initialize inf loop protect;
initialize which edges we have to go;
while length of edges to go > 0 do
    check inf loop protect and decrease it;
    obtain last edge with it's parent;
    put ok flag to False;
    if atom in at least in one ring then
        if number of rings is 1 then
            if this ring not in visited rings then
                make ring rule;
                append this rule to visited;
                put ok to True;
            end
        else if number of rings is 2 then
            find the invested ring;
            make ring rule;
            append this rule to visited;
            put ok to True;
    end
end
if ok is True then
    make simple linear rule;
end
save rule;
append to edges we have to go the last element with rule;
```

Decode algorithm is pretty much the same:

Algorithm 2: Decode algorithm

Result: Array of numbers, which represent rules

Input : Array with encoded numbers

Output: RDKit molecule

foreach *rule* $\in rules$ **do**

  if *rules are combinable* **then**

    combine them;

  end

  make RDKit molecule from graph;

end

We consider, that rules could be combined if they have suitable nodes. This node has to satisfy a few specifications:

1. left node has to be non-terminal, right node have to be terminal
2. nodes could be incoming and outcoming. Only outcoming rules could be matched with incoming.
3. nodes have to satisfy valency rules. A number of incoming bonds have to satisfy the number of outcoming rules.

Of course, a number of sites could be more than one. In this case, all these sites have to satisfy some hash functions. There are many hash functions, but we have to choose the simplest one. We make our artificial fingerprints, consisting of valencies and type of node. So, after applying this function we have rewrite the graph.

According to the definition of graph grammars, which rules have to satisfy insert or delete a pattern, our rules are only supported insert operation. This happens due to chemical science constraints, because we consider only molecules, no salts, or something like this. This is very interesting, to consider these molecular grammars on transformable systems and apply this technique to reactions.

One important, which have to be heard. When we encode our molecule from the graph or to graph we have to compare these rules according to graph equality. We take that parts are equal if they are isomorphic. Isomorphism of graphs G and H is a bijection between the vertex sets of G and H:

$$f : V(G) \to V(H)$$

such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H. Graphs are called isomorphic is an isomorphism exists between two graphs.

All this theory actually not be so good if it hasn't got a practical application. And, actually, it has. We have implemented all this stuff, and it's name - LegoGram. LegoGram is a Python package, which could be easily implemented in any recurrent neural network. Also, it has its own 100% sampler and mutator.

3.4 Optimization LegoGram

Our aim is to solve the smallest grammar problem. The smallest grammar problem is the problem of finding the smallest context-free grammar that generates a given string of characters. The problem statement is ambiguous and one has to define what is the size of grammar. One researcher assume, that the size of grammar is the number of rules on the right side of production rules. Other researchers add to this number of rules in this grammar. We assume that size of grammar is less than the original one.

Optimization of grammar is an NP-complete problem. NP-completeness problems require exponential time to consider all possible subproblems. For example, when we encode the 250k dataset, we obtain 3000 rules. Simple calculation:

$$e^{3000} \approx 10^{1302},$$

so it is too much. We haven't got enough time to consider all steps.

There are many ways to solve NP-complete problems. The popular ones are greedy algorithms and dynamic programming. We will consider the greedy approach. After encoding we obtain a number of rules, so each molecule could be considered as an array of a number of rules. We can reduce this task to the maximum substring problem. But actually, we can't!!! We didn't take into account the rules of our grammar. So, if we combine substring of five terminals - sequence of carbons - there will not sense in it, without rules with non-terminals. We can't add these rules from non-terminals. Thus, we obtain a constraint. All constructed rules have to be squeezed from a molecule and inserted back. The idea is as simple as a Lego. We could construct only those rules, which could be deleted from f.e. one castle and inserted into another. And this is the point!

So, the algorithm is very simple. After encoding these molecules we split each array to subarray with this constraint: each subarray has to be construable. We have to have the ability to construct one rule from this subset. A simple example to encode benzene we have to obtain seven rules: one rule consists of six non-terminals (some kind places) and six terminals. Of course, in different parts of molecules, five or six carbon terminals will appear very frequently. But these five non-terminals will have no sense without this one rule with six non-terminals. So, if we want to construct rules we have to add this non-terminal rule. Partly, it is explaining the number of rules in

the dataset: we actually have to preserve information about incoming and outcoming parts.

Algorithm 3: Greedy optimizer

```
Result: Constructed rules
Input  : Encoded molecules: 2d array, number of rules to construct n
Output: Constructed rules: arrays
foreach molecule ∈ molecules do
    encoded molecule ← [];
    current ← first rule;
    rules combined ← [first rule];
    foreach rule ∈ rules do
        if rule could be combined with current then
            current ← rule;
            rules combined ← rule;
        else
            encoded molecule ← rules combined;
            current ← rule;
            rules combined ← [rule];
        end
    end
end
count all subrules;
add more frequent n constructed rules;
```

This works well. Actually, there are many other approaches, for example, we could construct the longest rules or optimize rules for a subset of molecules. But our aim is to decrease the mean length of molecules. Another option is to optimize molecular grammars via neural networks, but for real problems, this approach is too complex and requires additional attention.

Chapter 4

Experimental part

Actual text and code published on https://github.com/FCreate/SkoltechDiplomaFinal.

4.1 Introduction

The experimental part has been divided into three parts. The first part is the application of molecular grammar to the 250k dataset and optimizing it. We will see, how LegoGram actually works and how it optimizes molecules. The second part is about applying LegoGram to recurrent neural networks approach. We have considered three types of models: vanilla recurrent neural network, reinforcement learning approach, and variational autoencoder.

In this work, we use a 250k dataset. It is a very famous dataset and it has been used in most papers considering application machine learning in generation tasks. 250k dataset is the part of very famous ZINC dataset. ZINC database is a free database of commercially-available compounds for virtual screening. ZINC contains over 230 million purchasable compounds in ready-to-dock, 3D formats. ZINC also contains over 750 million purchasable compounds.

Standardization of molecules is the process that reduce molecules to unified form. We have used the MolVS standardization tool. It has normalization of functional groups to a consistent format, recombination of separated charges, standardization of removal of stereochemistry information, and some other. Another preprocessing procedure that we have used is canonization. It's all about representing SMILES string in some uniform view, thus the same molecules will have equal SMILES strings.

After preprocessing of this dataset we have obtained 249456 unique SMILES strings. The average length of strings is 44.3 symbols.

As one can see distribution close to normal, so we expect better compression after using our grammar.

After we have applied our grammar to the current dataset, we obtain some compression. A very important result on this stage is that our grammar encodes and decodes molecules with 100% quality. Also, we have obtained 100% validity after augmentation. Augmentation is the process to produce synthetic data for training. In many task, augmentation could outperform model which have trained on not augmented data. We have to augment data for chemoinformatics that could do

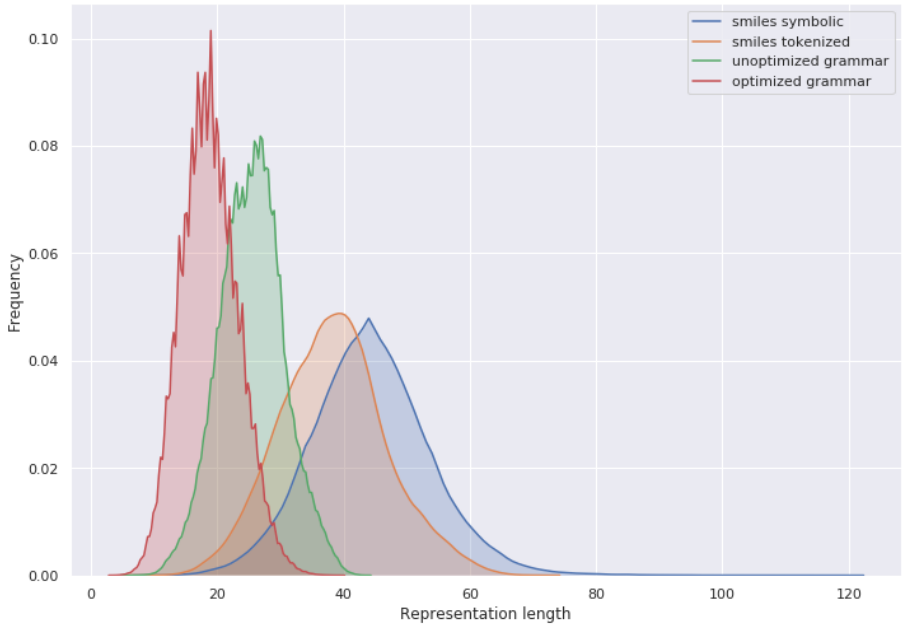

Figure 4.1: Histogram of original, encoded and optimized SMILES

implemented as making not canonical SMILES strings from canonical. So, for a human it's the same molecules but for machine learning algorithms it's not the same data.

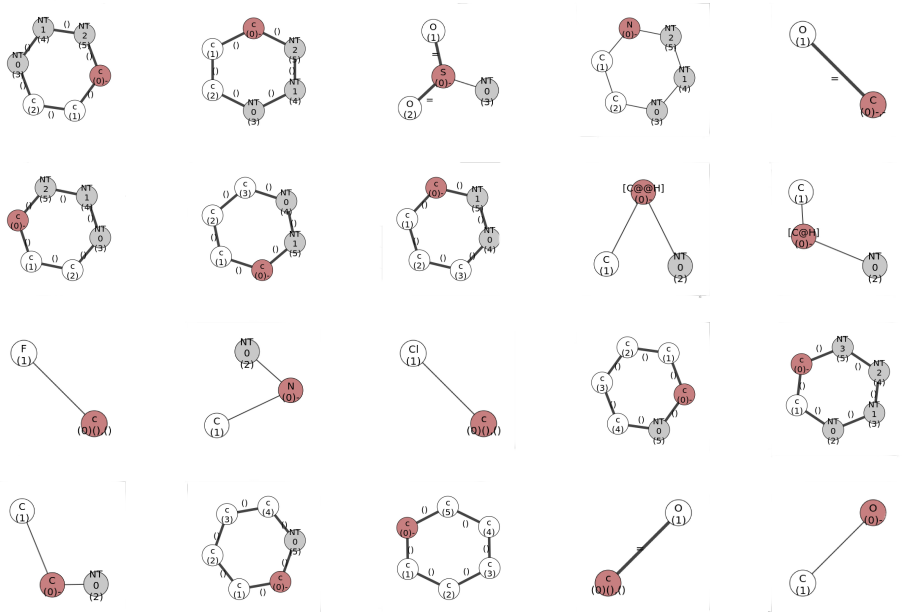

Figure 4.2: Example of optimized grammar blocks

After applying LegoGram to molecules we have decreased the length of molecules and distribution become more sprawling. It takes place because encoded molecules haven't not equal length in different position of cycles and other structural properties. Another one good property is

the speed of the encoding and decoding. The dataset containing approximately 250000 molecules has been encoded for approximately 15 minutes on a simple laptop with an i5 processor.

4.2 *De-novo* generation of molecules

QSAR inverse problem

After obtaining these rules one could ask: Why have we calculated them? What advantages we will obtain? So, we have solved two main issues. The first is generating invalid molecules. Many techniques for generating molecular sequence have borrowed from fast-growing part of deep learning techniques: natural language processing. But the most important feature of these methods is evaluating their result by a human. So, if the algorithm, for example, for a generation has failed in one-word humans could understand the sense by context. In chemoinformatics field generated output validated not only by humans but by computer. RDkit is a python package for chemoinformatics. In this package, one could validate molecules simply by rules of chemistry. We consider that molecule is valid if a molecule passes through the RDKit validation tool. It is good for some tasks, but there are many molecules that haven't chemical sense but could pass through RDKit, for example, CCCCCC - simply sequence of carbons with hydrogens (part of polyethylene). So, validation of molecules in chemoinformatics is no a simple task, but we consider that molecule is valid if it could pass through the RDKit validation tool.

One could evaluate generation algorithms by many parameters, but the most important is how many generated molecules are valid, how many unique molecules we obtain through generation. Another important property of generation for variational autoencoders is how many molecules could be correctly reconstructed. Another option are to estimate the loglikelihood of loss functions or something else parameters of a neural network, but this loss is synthetic and not reflect the real situation.

4.2.1 Recurrent neural network

For the first experiment, we have used three-layer GRU. GRU was used because it trains faster with producing the same results. The aim of this task is to compare our method of molecular grammars with basic RNN on SMILES string. We expect that this method will provide the same level of quality as state-of-the-art methods. The first plot is the plot of training. It is a plot in terms of loss vs epochs. As you can see SMILES and Grammar models have produced the same result, but our approach always generates 100% valid molecules. Of, course it is training faster in terms of valid molecules per number of steps because we obtain 100% valid almost instantly.

Some generated molecules have plotted on the figures for SMILES model A.1 and grammar

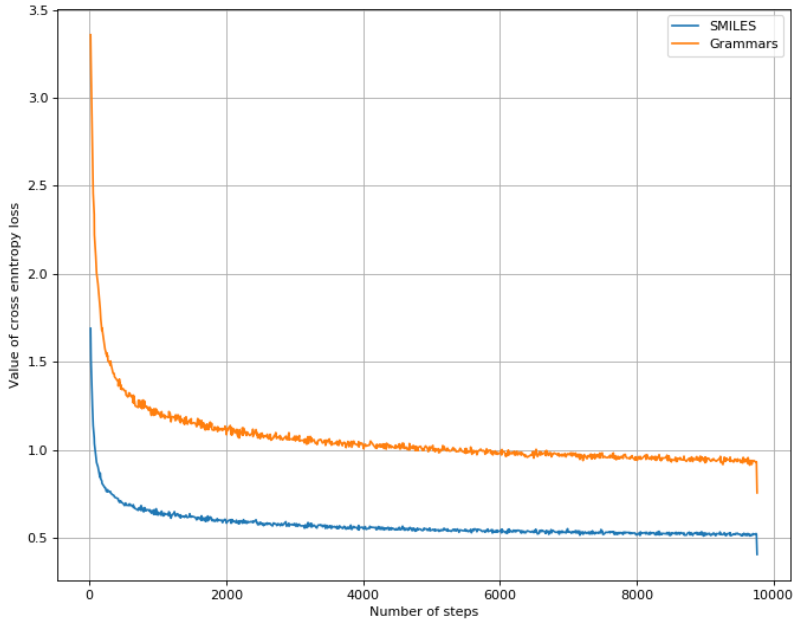

Figure 4.3: Cross Entropy loss for SMILES and Grammars

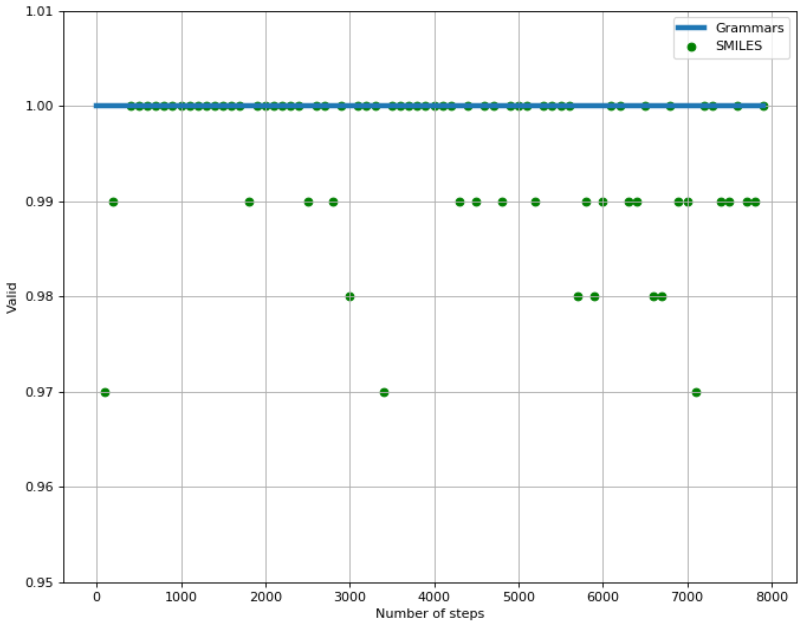

Figure 4.4: Valid molecules which have generated in SMILES and Grammar notations

model A.2. As you can see, molecules generated by Grammar much more complicated. This happens because building blocks of this grammar, as a rule, are bigger than the basic SMILES model. So, the molecular grammar model could produce a valid molecule bigger than the SMILES model.

Reinforcement learning

This grammar has many applications. One of these is reinforcement learning. We have trained neural network which is equal to https://github.com/MarcusOlivecrona/REINVENT project. For simplicity we have used no sulfur molecular function, so we want to obtain molecules without sulfur. Both SMILES and Grammar model are correct, but the grammar model has achieved result much faster due to blocks. Neural networks learns which blocks have no sulfur and use it.

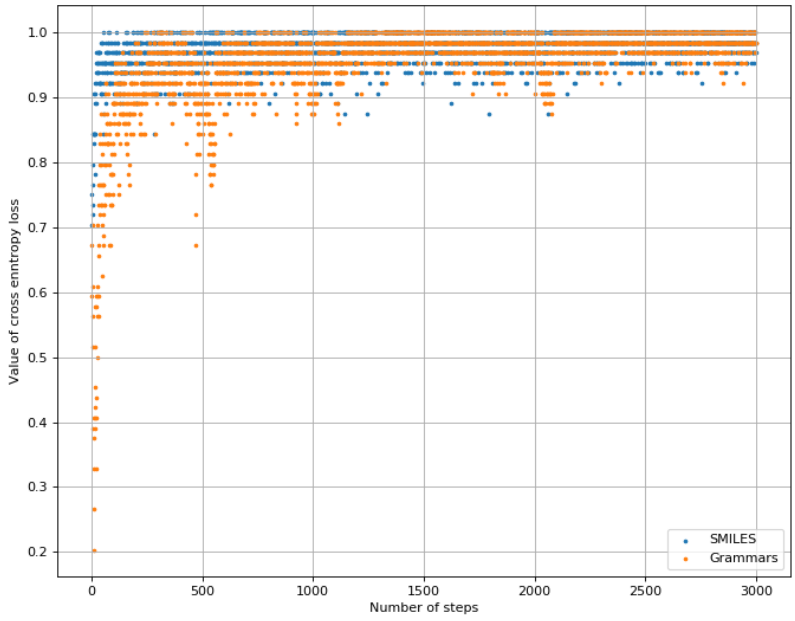

Figure 4.5: Score for REINVENT SMILES and Grammar

This grammar has many applications. One of these is reinforcement learning. We have trained neural network which is equal to https://github.com/MarcusOlivecrona/REINVENT project. For simplicity we have used no sulfur molecular function, so we want to obtain molecules without sulfur. Both SMILES and grammar model are correct, but the grammar model has achieved result much faster due to blocks. Neural networks learns which blocks have no sulfur and use it.

Example of molecules are shown on figure A.3 for SMILES model and for grammar model: A.4.

On this easy task, we have proved the correctness of our method. But we could consider our method on the really difficult problems: generate drug-like molecules with a condition - boiling point should be greater than some value (in our experiments: 300 degrees by Celsius) also molecules have to satisfy drug-like condition. We have combined Lipinksi's rule of five and some other rules:

- 160 <= molecular weight <= 480

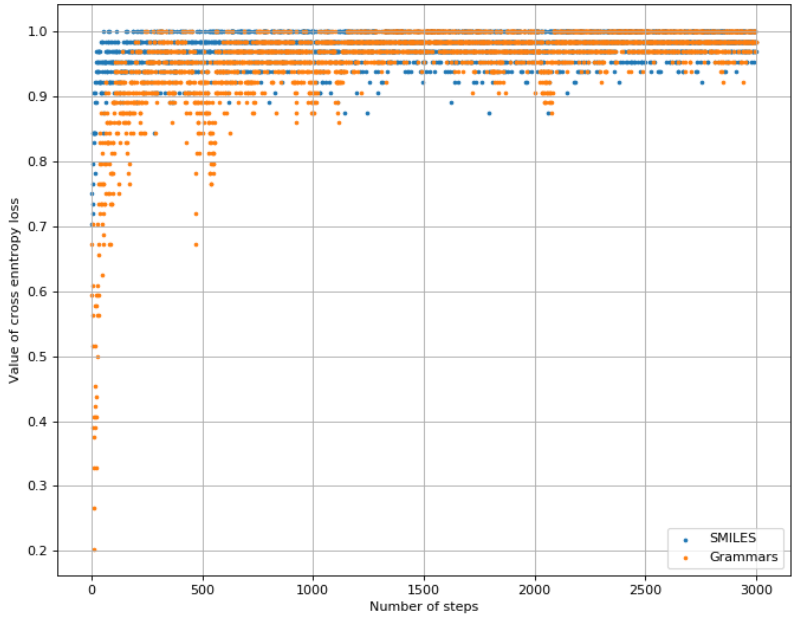

Figure 4.6: Score for REINVENT SMILES and Grammar

- -0.4 <= logp <= 5.6
- 20 <= number of atoms <= 70
- 40 <= molar reflactivity <= 130
- bioconcentration factor < 3
- developmental toxicity should be Negative
- number of rings > 0
- number of acceptors <= 10
- number of donors <= 5

Every true rule from this has given 1 point to scoring function and our desired property gives 5 points. The distribution is presented on the picture. Examples of molecules are shown on figure A.5.

In other case - generating molecules with boiling point equal 300, we have calculated property. We have evaluated our reinforcement learning model on the basic machine learning model (XGBoost) trained on the TEST database. Obtained RMSE is 15.6 and that is a good result.

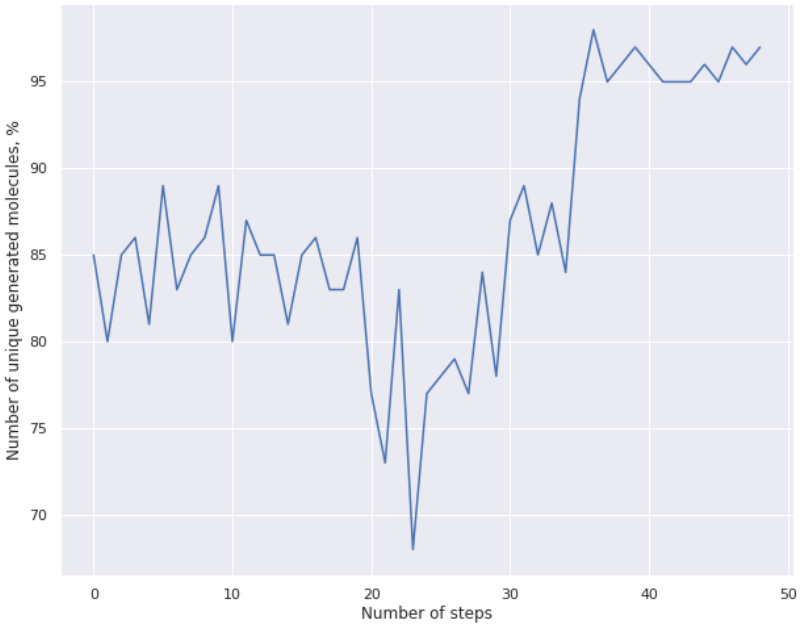

Figure 4.7: Percent of unique generated SMILES

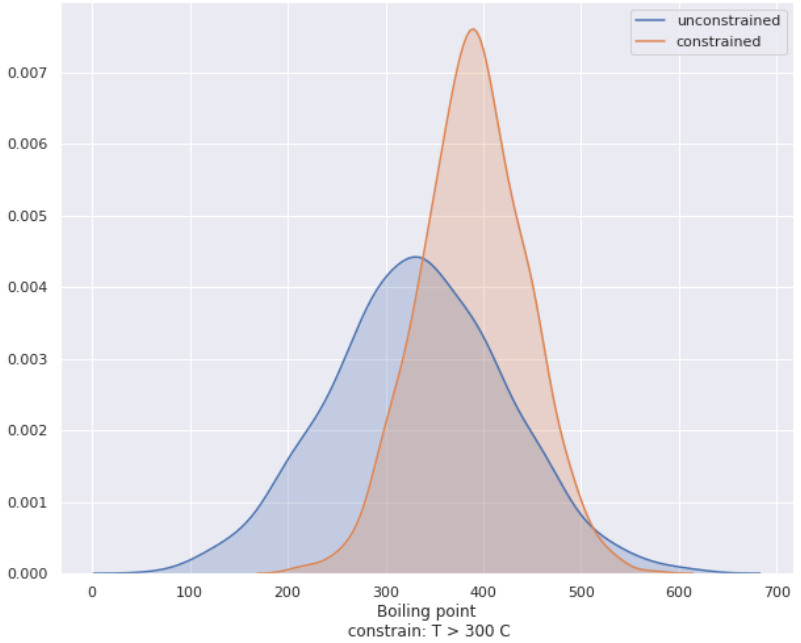

Figure 4.8: Boiling point distribution

Variational autoencoder

Recurrent neural networks and reinforcement learning are good ones, but the most important one is the variational autoencoder technique. There are many ways to produce molecules with desired

properties, variational autoencoder is one of them. But actually, variational autoencoder is a much more powerful technique, then simply generation molecules with desired properties, it actually encodes all chemical space to some latent. This technique allows to interpolating from one molecule to another through this latent space.

The main advantage of our method is that it can generate 100% valid molecules. Generated molecules will not only be semantic correct, but they will be syntactic correct also and have some chemical sense. This is very important for variational autoencoders, because first studies of VAE in chemoinformatics produce a low number of correct molecules.

We have trained two types of variational autoencoders on the 250k dataset. We have chosen 3 layer encoder and decoder GRU networks and trained them using Adam optimizer with an initial learning rate $\eta = 10^{-4}$. Also, how it done in other works, considering variational autoencoder we have varied proportion of cross-entropy loss and KL-divergence during the training by sigmoid rule. Our result for variational autoencoder is presented in the table:

Method	Reconstructed %	Valid prior %
CVAE	44.6	0.7
GVAE	53.7	7.2
JT-VAE	76.7	100
MHG	94.8	100
Ours	92.7	100

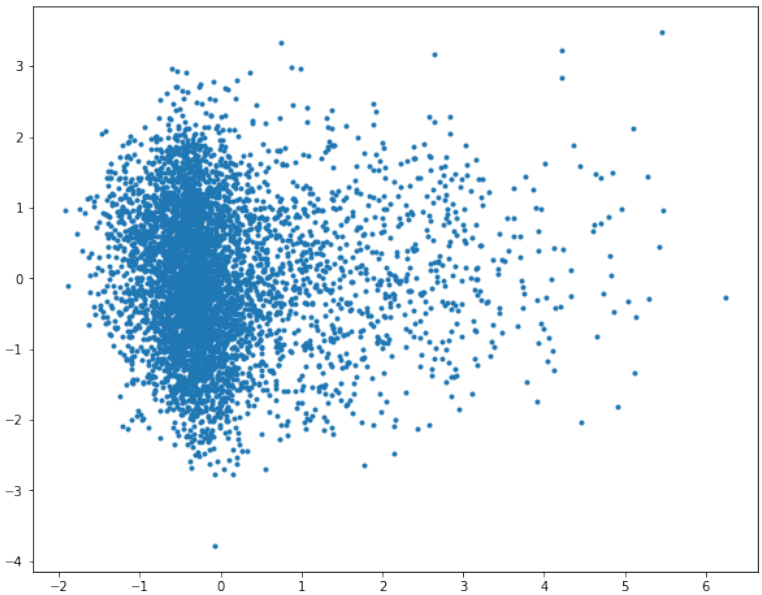

Figure 4.9: Latent space for LegoGram variational autoencoder

Examples of generated molecules are shown on A.6.

Chapter 5

Conclusion

The inverse QSAR problem is the major challenge for drug discovery. The successful, solutions to this problem leads us close to the drug-discovery fully computational pipeline. Here in this work, we propose a new method for generating molecular structures by deep learning. Our method allows generating valid chemical structures, which are satisfy desired chemical properties. It opens doors to sample drug-candidates from chemical space directly.

The graph could be designed in different ways, one way is graph grammar. Graph grammars have to support insert (and delete) operations. In this paper, a new grammar for constructing molecules was considered. This problem is actual today and papers have been published on this topic. Now, the state-of-the-art solution is molecular hypergraph grammar, which produces 100% semantic correct molecules. Nevertheless, this method is very difficult and complicated in terms of computing resources.

We have designed another method - molecular, which is based on node-labeled controlled graph grammar. It is as simple as Lego. Also, we have implemented a Python package. This package provides a separate abstraction layer and can be used in any type of recurrent neural network and different generative approaches, i. e. variational autoencoder, reinforcement learning. We experimentally proved that it can encode and decode any type of molecules. Also, at the inference stage, our algorithm produces 100% semantically correct molecules.

We validated our approach in three neural settings: i) generative unrestricted recurrent neural network, ii) reinforcement learning and iii) variational autoencoder. We demonstrated that unrestricted generation RNNs have a similar performance as the SMILES model, but converges much faster on at the beginning. Reinforcement learning model works reasonably well on SMILES, but actually it provides comparable performance with our LegoGram model on a graph model. We have evaluated the efficiency of RL-model on two tasks. The first one is the generation of the structures with additional chemical restriction: i.e. generation of molecules without sulphur atom. Another one is the generation drug-like structures with a boiling point greater than 300. For both models, the results were impressively good. No-sulphur model produced 100% of molecules without sulphur atoms. In constrained optimization - generating drug-like molecules with boiling point 300, we have achieved RMSE 15.6, which is comparable with forwarding model error. Another optimization, i.e. generation drug-likeness molecules property - boiling point greater than 300. In

this task we have designed drug-like molecules, with distribution mean is 389 and the standard deviation is 54. Also, we have evaluated this grammar on variational autoencoder. Variational autoencoder model produces 100% prior valid molecules and 92.7% molecules are reconstructed correctly.

Another part of our study is optimizing molecular grammar. We have proposed algorithm for optimizing this grammar on any dataset as you wish. This optimization has reduced mean length of molecules for approximately 25% by adding 3% constructed rules. A promising way for Legogram is compressing extra-large chemical datasets. So, for big datasets, our grammar could be used for compressing data.

We have regarded LegoGram as a universal tool for chemoinformatics. Despite the already discussed applications, we believe that our library could inspire new ways for solving existing chemical problems.

Appendix A

Figures

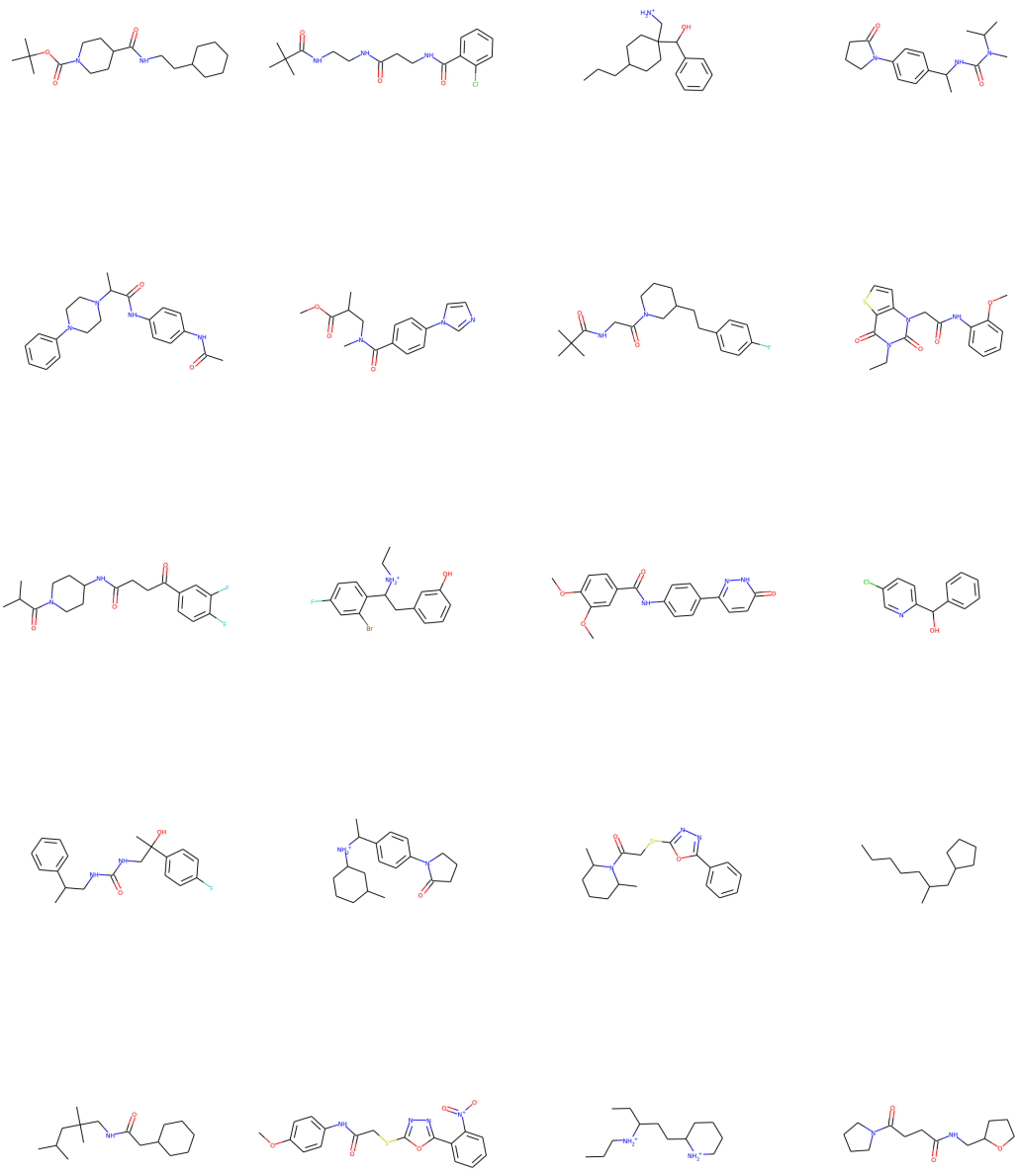

Figure A.1: Generated molecules for SMILES notation

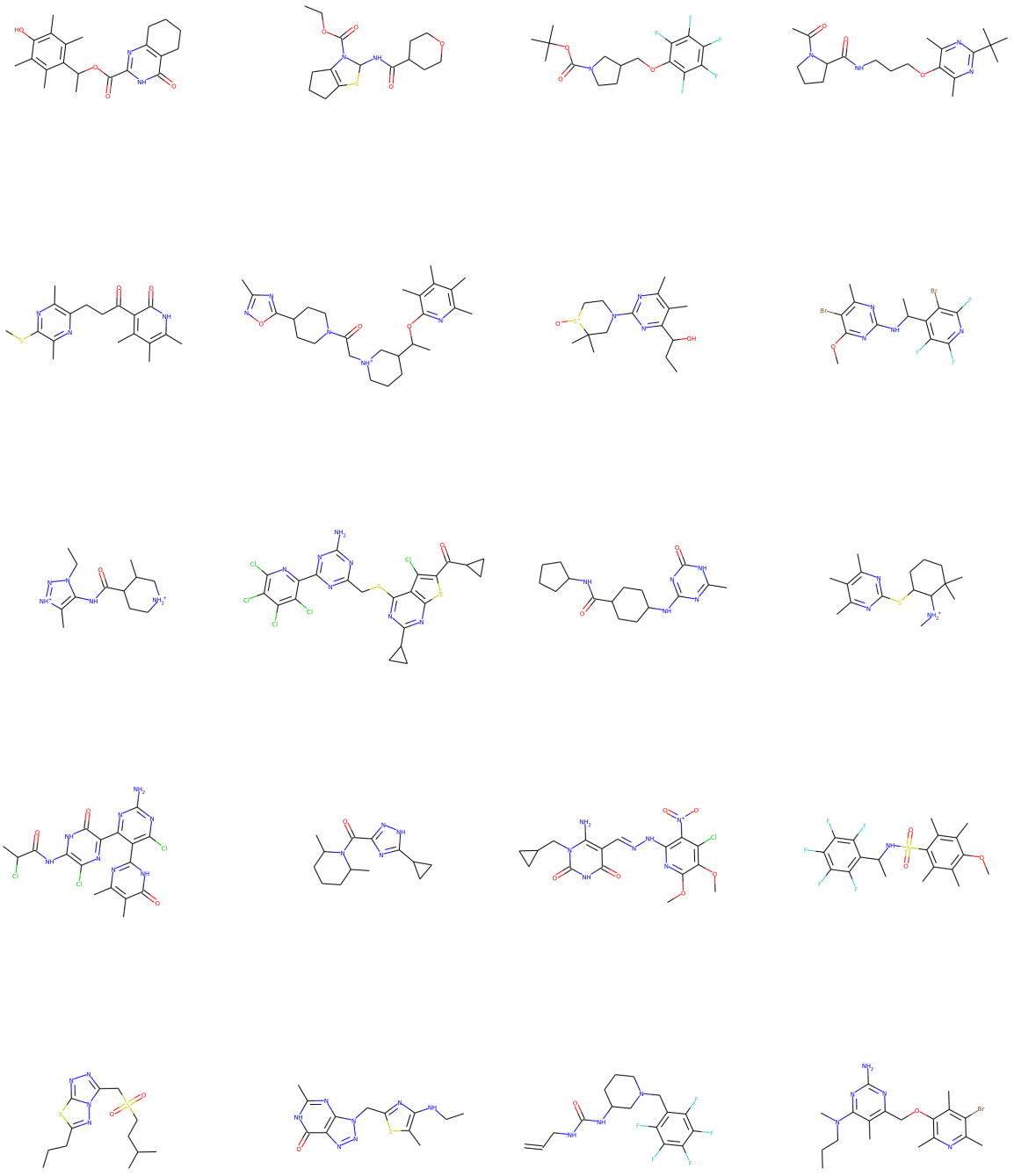

Figure A.2: Generated molecules for grammar notation

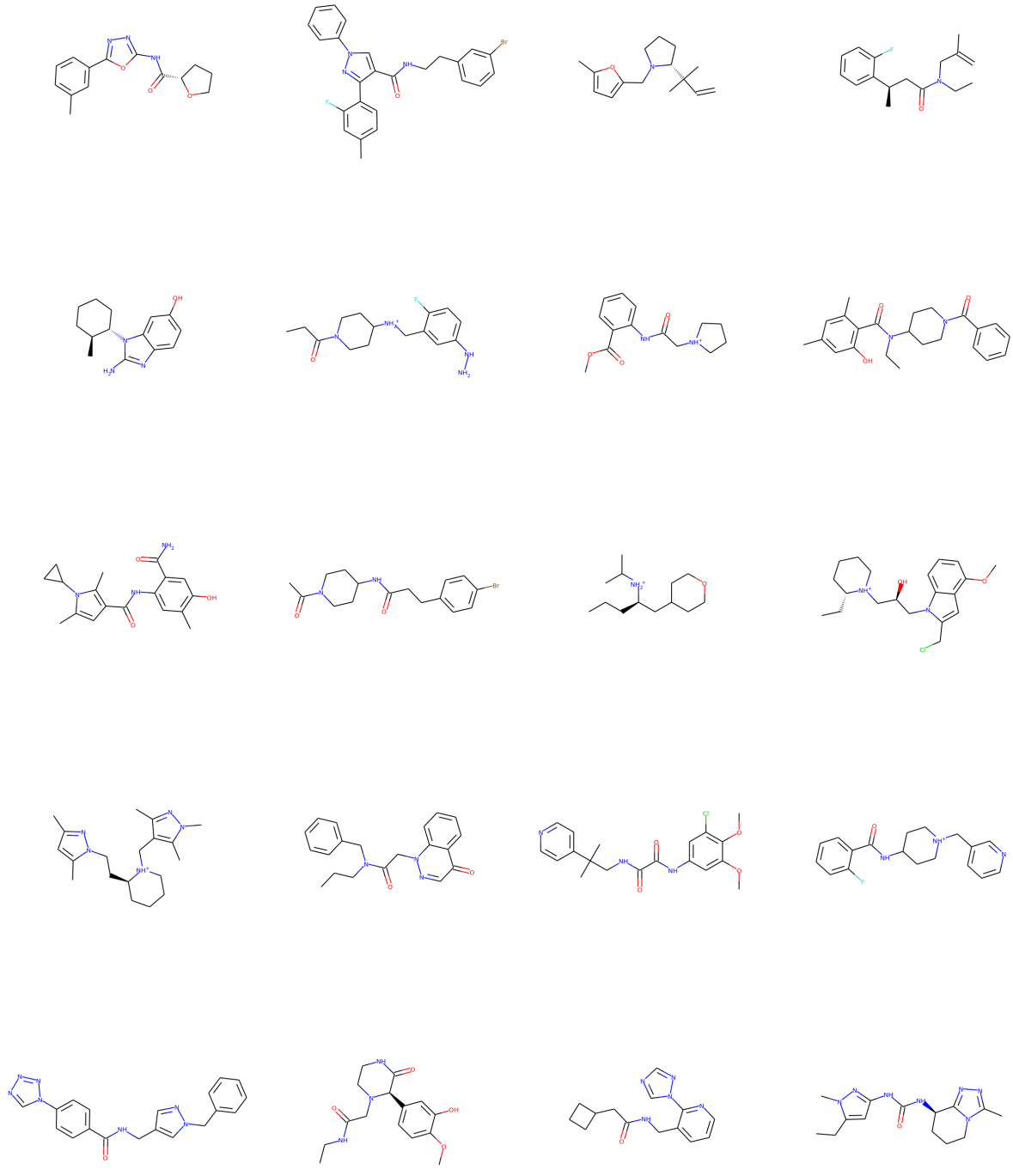

Figure A.3: Example of generated no sulfur molecules via SMILES model

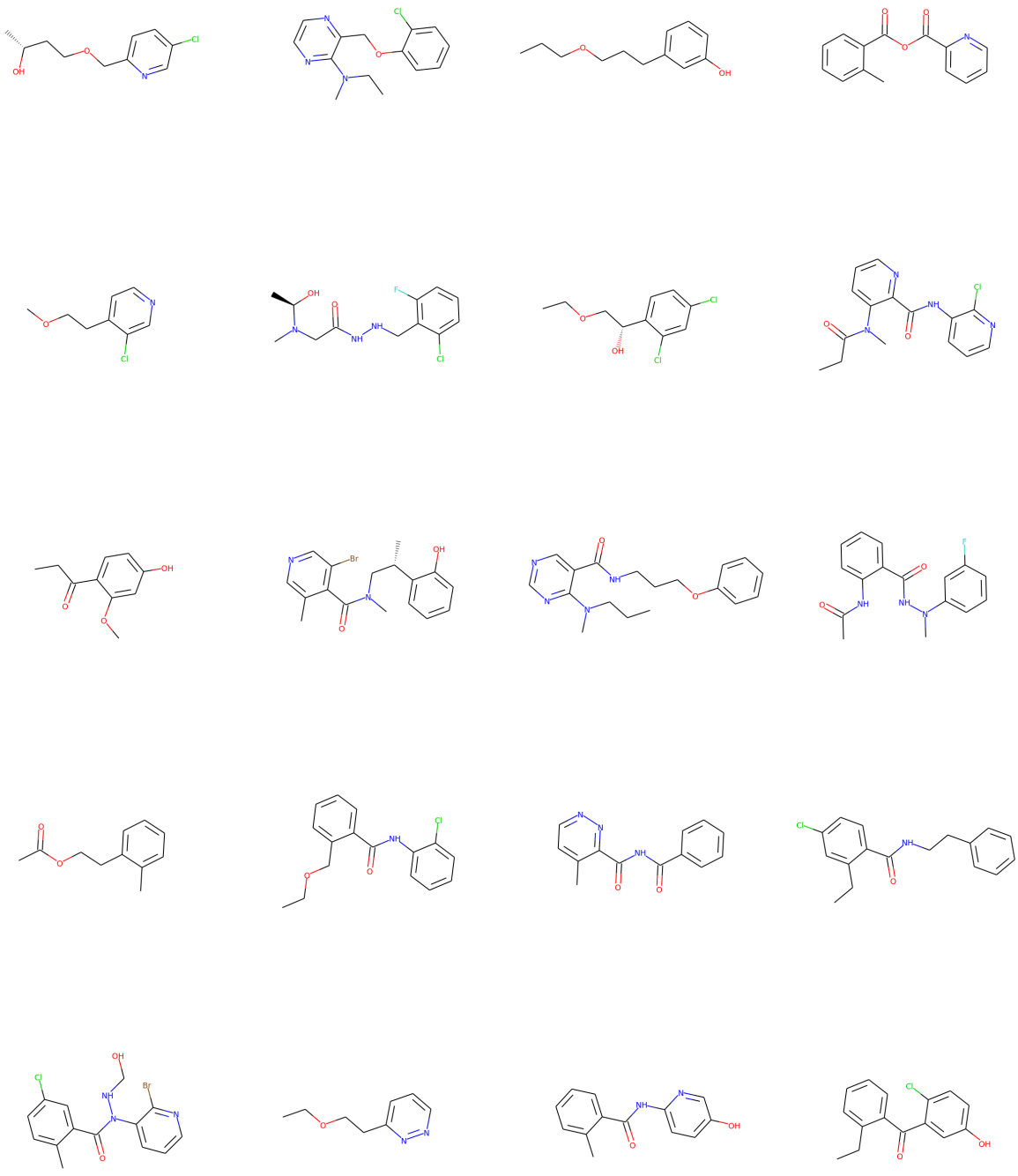

Figure A.4: Example of generated no sulfur molecules via Grammar model

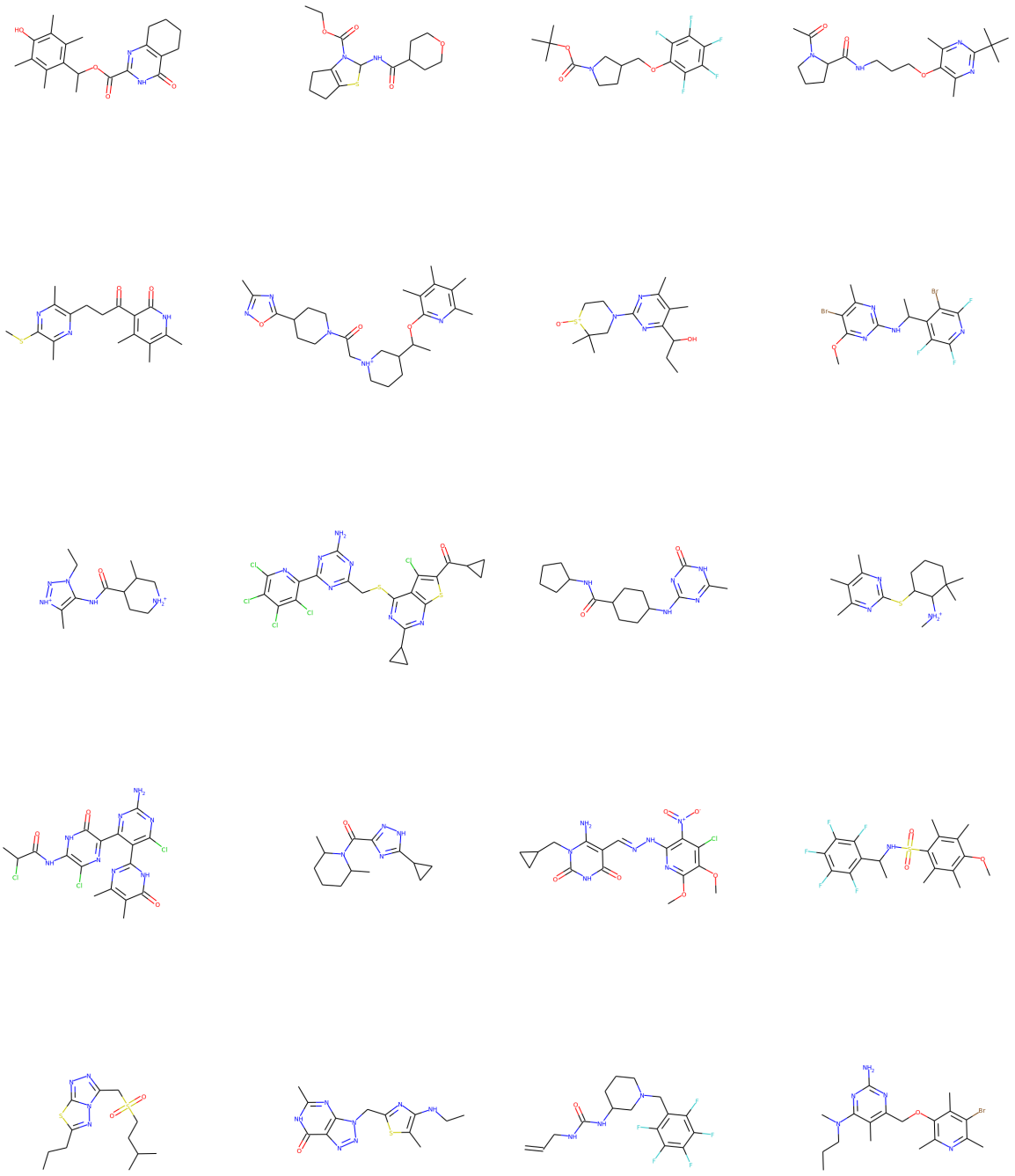

Figure A.5: Generated molecules on reinforcement learning for boiling point more then 300

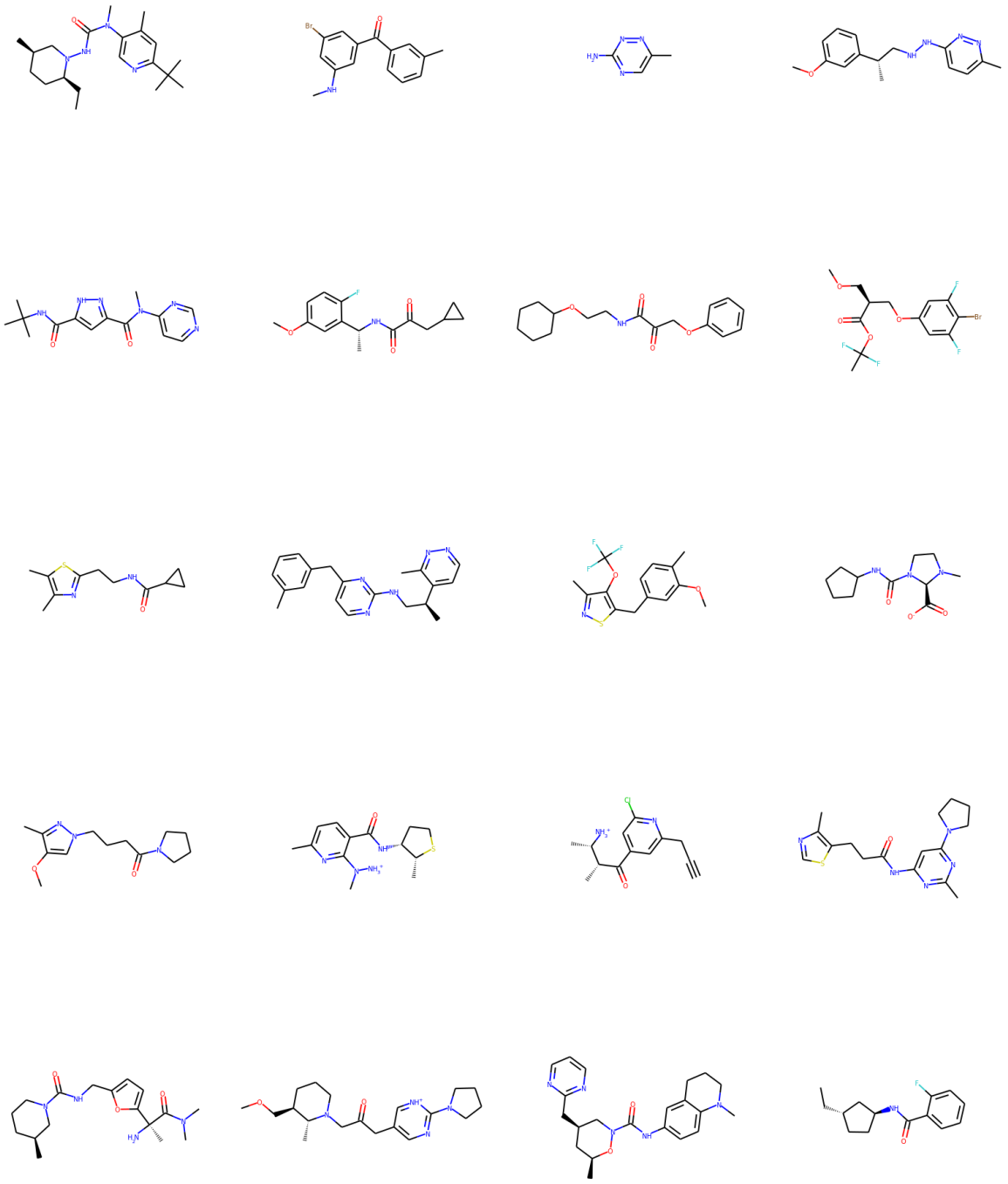

Figure A.6: Example of generated molecules via Grammar model on Variational Autoencoder

Bibliography

[Agnese et al., 2020] Agnese, J., Herrera, J., Tao, H., and Zhu, X. (2020). A survey and taxonomy of adversarial neural networks for text□to□image synthesis. *WIREs Data Mining and Knowledge Discovery*.

[Al-Labadi et al., 2020] Al-Labadi, L., Patel, V., Vakiloroayaei, K., and Wan, C. (2020). Kullback–leibler divergence for bayesian nonparametric model checking. *Journal of the Korean Statistical Society*.

[Ardizzone et al., 2020] Ardizzone, L., Mackowiak, R., Rother, C., and Köthe, U. (2020). Exact information bottleneck with invertible neural networks: Getting the best of discriminative and generative modeling.

[Arora and Doshi, 2018] Arora, S. and Doshi, P. (2018). A survey of inverse reinforcement learning: Challenges, methods and progress.

[Arulkumaran et al., 2017] Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

[Arvinte et al., 2019] Arvinte, M., Tewfik, A. H., and Vishwanath, S. (2019). Deep log-likelihood ratio quantization. *2019 27th European Signal Processing Conference (EUSIPCO)*.

[Bielska et al., 2011] Bielska, E., Lucas, X., Czerwoniec, A., M. Kasprzak, J., H. Kaminska, K., and M. Bujnicki, J. (2011). Review paper
virtual screening strategies in drug design – methods and applications. *BioTechnologia*, 92(3):249–264.

[Buhrmester et al., 2019] Buhrmester, V., Münch, D., and Arens, M. (2019). Analysis of explainers of black box deep neural networks for computer vision: A survey.

[Bulinski and Dimitrov, 2019] Bulinski, A. and Dimitrov, D. (2019). Statistical estimation of the kullback-leibler divergence.

[Byers et al., 1990] Byers, J. A., Birgersson, G., L□fqvist, J., Appelgren, M., and Bergstr□m, G. (1990). Isolation of pheromone synergists of bark beetle, pityogenes chalcographus, from complex insect-plant odors by fractionation and subtractive-combination bioassay. *Journal of Chemical Ecology*, 16(3):861–876.

[Chaudhari et al., 2019] Chaudhari, S., Polatkan, G., Ramanath, R., and Mithal, V. (2019). An attentive survey of attention models.

[Che et al., 2018] Che, Z., Purushotham, S., Cho, K., Sontag, D., and Liu, Y. (2018). Recurrent neural networks for multivariate time series with missing values. *Scientific Reports*, 8(1).

[Cho et al., 2014] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[Chung et al., 2014] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling.

[Chung et al., 2015] Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A., and Bengio, Y. (2015). A recurrent latent variable model for sequential data.

[Coffin and Smith,] Coffin, D. and Smith, R. E. Linkage learning in estimation of distribution algorithms. In *Studies in Computational Intelligence*, pages 141–156. Springer Berlin Heidelberg.

[Congreve et al., 2003] Congreve, M., Carr, R., Murray, C., and Jhoti, H. (2003). A 'rule of three' for fragment-based lead discovery? *Drug Discovery Today*, 8(19):876–877.

[Cowan, 1989] Cowan, J. D. (1989). Neural networks: The early days. In *NIPS*.

[Demir et al., 2019] Demir, S., Mutlu, U., and Özgur Özdemir (2019). Neural academic paper generation.

[Deutsch, 2018] Deutsch, L. (2018). Generating neural networks with neural networks.

[Ding and Gimpel, 2019] Ding, X. and Gimpel, K. (2019). Latent-variable generative models for data-efficient text classification. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

[Doersch, 2016] Doersch, C. (2016). Tutorial on variational autoencoders.

[Dubois and Mouzon, 2014] Dubois, P. and Mouzon, O. D. (2014). Market Size and Pharmaceutical Innovation.

[Echegoyen et al., 2013] Echegoyen, C., Mendiburu, A., Santana, R., and Lozano, J. A. (2013). On the taxonomy of optimization problems under estimation of distribution algorithms. *Evolutionary Computation*, 21(3):471–495.

[Elton et al., 2019] Elton, D. C., Boukouvalas, Z., Fuge, M. D., and Chung, P. W. (2019). Deep learning for molecular design—a review of the state of the art. *Molecular Systems Design Engineering*, 4(4):828–849.

[Gao and Coley, 2020] Gao, W. and Coley, C. W. (2020). The synthesizability of molecules proposed by generative models. *Journal of Chemical Information and Modeling*.

[Ghose et al., 1999] Ghose, A. K., Viswanadhan, V. N., and Wendoloski, J. J. (1999). A knowledge-based approach in designing combinatorial or medicinal chemistry libraries for drug discovery. 1. a qualitative and quantitative characterization of known drug databases. *Journal of Combinatorial Chemistry*, 1(1):55–68.

[Ghosh et al., 2019] Ghosh, P., Sajjadi, M. S. M., Vergari, A., Black, M., and Schölkopf, B. (2019). From variational to deterministic autoencoders.

[Giri et al., 2016] Giri, E. P., Fanany, M. I., and Arymurthy, A. M. (2016). Combining generative and discriminative neural networks for sleep stages classification.

[Goh et al., 2014] Goh, G. B., Hodas, N. O., and Vishnu, A. (2014). Deep Learning for Computational Chemistry. pages 1–50.

[Gómez-Bombarelli et al., 2018] Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. (2018). Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. *ACS Central Science*, 4(2):268–276.

[Graves, 2013] Graves, A. (2013). Generating sequences with recurrent neural networks.

[Gregor et al., 2015] Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., and Wierstra, D. (2015). Draw: A recurrent neural network for image generation.

[Grigorescu, 2019] Grigorescu, S. (2019). A Survey of Deep Learning Techniques for Autonomous Driving.

[Guo et al., 2019] Guo, T., Lin, T., and Antulov-Fantulin, N. (2019). Exploring interpretable lstm neural networks over multi-variable data.

[Harik et al., 2006] Harik, G. R., Lobo, F. G., and Sastry, K. (2006). Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA). In *Scalable Optimization via Probabilistic Modeling*, pages 39–61. Springer Berlin Heidelberg.

[Hopfield, 1982] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558.

[HOSKINS, 1991] HOSKINS, B. J. (1991). Towards a pv-θ view of the general circulation. *Tellus A*, 43(4):27–35.

[Ji and Shen, 2019] Ji, C. and Shen, H. (2019). Stochastic variational inference via upper bound.

[Jin et al., 2018] Jin, W., Barzilay, R., and Jaakkola, T. (2018). Junction Tree Variational Autoencoder for Molecular Graph Generation.

[Jin et al., 2020] Jin, W., Barzilay, R., and Jaakkola, T. (2020). Multi-objective molecule generation using interpretable substructures.

[Jing and Xu, 2019] Jing, K. and Xu, J. (2019). A survey on neural network language models.

[Jr, 2019] Jr, J. F. R. (2019). A survey on Big Data and Machine Learning for Chemistry. pages 1–48.

[Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

[Kajino, 2019] Kajino, H. (2019). Molecular Hypergraph Grammar with Its Application to Molecular Optimization. (2):1–19.

[Kar and Haldar, 2016] Kar, R. and Haldar, R. (2016). Applying chatbots to the internet of things: Opportunities and architectural elements. *International Journal of Advanced Computer Science and Applications*, 7(11).

[Kim et al., 2018] Kim, Y., Wiseman, S., and Rush, A. M. (2018). A tutorial on deep latent variable models of natural language.

[Kingma and Welling, 2019] Kingma, D. P. and Welling, M. (2019). An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392.

[Krusinga et al., 2019] Krusinga, R., Shah, S., Zwicker, M., Goldstein, T., and Jacobs, D. (2019). Understanding the (un)interpretability of natural image distributions using generative models.

[Kusner et al., 2017] Kusner, M. J., Paige, B., and Hemández-Lobato, J. M. (2017). Grammar variational autoencoder. *34th International Conference on Machine Learning, ICML 2017*, 4:3072–3084.

[Lecun, 2001] Lecun, Y. (2001). A theoretical framework for back-propagation.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Lee et al., 2019] Lee, J., Chirkov, N., Ignasheva, E., Pisarchyk, Y., Shieh, M., Riccardi, F., Sarokin, R., Kulik, A., and Grundmann, M. (2019). On-device neural net inference with mobile gpus. *CoRR*, abs/1907.01989.

[Leeson and Springthorpe, 2007] Leeson, P. D. and Springthorpe, B. (2007). The influence of drug-like concepts on decision-making in medicinal chemistry. *Nature Reviews Drug Discovery*, 6(11):881–890.

[Leo et al., 1971] Leo, A., Hansch, C., and Elkins, D. (1971). Partition coefficients and their uses. *Chemical Reviews*, 71(6):525–616.

[Leong et al., 2014] Leong, M., Ng, P. K., Jee, K., Yue Hang, T., and Kai Shen, D. L. (2014). A review of china's technological developments in the 20th century.

[Li and Malik, 2018] Li, K. and Malik, J. (2018). Implicit maximum likelihood estimation.

[Li et al., 2020] Li, Z., Yang, W., Peng, S., and Liu, F. (2020). A survey of convolutional neural networks: Analysis, applications, and prospects.

[Lionta et al., 2014] Lionta, E., Spyrou, G., Vassilatis, D., and Cournia, Z. (2014). Structure-based virtual screening for drug discovery: Principles, applications and recent advances. *Current topics in medicinal chemistry*.

[Lipinski, 2004] Lipinski, C. A. (2004). Lead- and drug-like compounds: the rule-of-five revolution. *Drug Discovery Today: Technologies*, 1(4):337–341.

[Lipinski et al., 2001] Lipinski, C. A., Lombardo, F., Dominy, B. W., and Feeney, P. J. (2001). Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings 1pii of original article: S0169-409x(96)00423-1. the article was originally published in advanced drug delivery reviews 23 (1997) 3–25. 1. *Advanced Drug Delivery Reviews*, 46(1-3):3–26.

[Lipton et al., 2015] Lipton, Z. C., Berkowitz, J., and Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning.

[Livne et al., 2020] Livne, M., Swersky, K., and Fleet, D. J. (2020). Sentencemim: A latent variable language model.

[Lu, 2019] Lu, X. (2019). Learning to generate questions with adaptive copying neural networks. *Proceedings of the 2019 International Conference on Management of Data - SIGMOD '19*.

[Mangal et al., 2019] Mangal, S., Modak, R., and Joshi, P. (2019). Lstm based music generation system. *IARJSET*, 6(5):47–54.

[Mansimov et al., 2019] Mansimov, E., Mahmood, O., Kang, S., and Cho, K. (2019). Molecular geometry prediction using a deep generative graph neural network. *Scientific Reports*, 9(1).

[Masegosa et al., 2019] Masegosa, A. R., Cabañas, R., Langseth, H., Nielsen, T. D., and Salmerón, A. (2019). Probabilistic models with deep neural networks.

[Mélanie-Bécquet et al., 2015] Mélanie-Bécquet, F., Ferguth, J., Gruel, K., and Poibeau, T. (2015). Archaeology in the digital age: From paper to databases. *CoRR*, abs/1507.02021.

[Mnasri, 2019] Mnasri, M. (2019). Recent advances in conversational nlp : Towards the standardization of chatbot building.

[Neglur et al., 2005] Neglur, G., Grossman, R. L., and Liu, B. (2005). Assigning unique keys to chemical compounds for data integration: Some interesting counter examples. In *Lecture Notes in Computer Science*, pages 145–157. Springer Berlin Heidelberg.

[Nielsen, 2019] Nielsen, F. (2019). On the kullback-leibler divergence between location-scale densities.

[Odaibo, 2019] Odaibo, S. (2019). Tutorial: Deriving the standard variational autoencoder (vae) loss function.

[Olsen et al., 2019] Olsen, J. J. W., Christensen, P. E., Hansen, M. H., and Johansen, A. R. (2019). Autoencoding undirected molecular graphs with neural networks.

[Oprea et al., 2001] Oprea, T. I., Davis, A. M., Teague, S. J., and Leeson, P. D. (2001). Is there a difference between leads and drugs? a historical perspective. *Journal of Chemical Information and Computer Sciences*, 41(5):1308–1315.

[Ozbayoglu et al., 2020] Ozbayoglu, A. M., Gudelek, M. U., and Sezer, O. B. (2020). Deep learning for financial applications : A survey. *Applied Soft Computing*, 93:106384.

[Pascanu et al., 2012] Pascanu, R., Mikolov, T., and Bengio, Y. (2012). On the difficulty of training recurrent neural networks.

[Pham et al., 2018] Pham, T., Tran, T., and Venkatesh, S. (2018). Graph memory networks for molecular activity prediction. *2018 24th International Conference on Pattern Recognition (ICPR)*.

[Pope et al., 2018] Pope, P., Kolouri, S., Rostrami, M., Martin, C., and Hoffmann, H. (2018). Discovering molecular functional groups using graph convolutional neural networks.

[Popova et al., 2019] Popova, M., Shvets, M., Oliva, J., and Isayev, O. (2019). Molecularrnn: Generating realistic molecular graphs with optimized properties.

[Potdar et al., 2017] Potdar, K., Pardawala, T., and Pai, C. (2017). A comparative study of categorical variable encoding techniques for neural network classifiers. *International Journal of Computer Applications*, 175:7–9.

[Pölsterl and Wachinger, 2019] Pölsterl, S. and Wachinger, C. (2019). Likelihood-free inference and generation of molecular graphs.

[Raghu and Schmidt, 2020] Raghu, M. and Schmidt, E. (2020). A survey of deep learning for scientific discovery.

[Ribeiro et al., 2019] Ribeiro, A. H., Tiels, K., Aguirre, L. A., and Schön, T. B. (2019). Beyond exploding and vanishing gradients: analysing rnn training using attractors and smoothness.

[Roche, 2011] Roche, A. (2011). Em algorithm and variants: an informal tutorial.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386.

[Sadeghi et al., 2014] Sadeghi, J., Sadeghi, S., and Niaki, S. T. A. (2014). Optimizing a hybrid vendor-managed inventory and transportation problem with fuzzy demand: An improved particle swarm optimization algorithm. *Information Sciences*, 272:126–144.

[Sadowski et al., 2013] Sadowski, K. L., Bosman, P. A., and Thierens, D. (2013). On the usefulness of linkage processing for solving MAX-SAT. In *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference - GECCO '13*. ACM Press.

[Sawada et al., 2019] Sawada, Y., Morikawa, K., and Fujii, M. (2019). Study of deep generative models for inorganic chemical compositions.

[Shakhatreh et al., 2019] Shakhatreh, H., Sawalmeh, A. H., Al-Fuqaha, A., Dou, Z., Almaita, E., Khalil, I., Othman, N. S., Khreishah, A., and Guizani, M. (2019). Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges. *IEEE Access*, 7:48572–48634.

[Shao et al., 2019] Shao, K., Tang, Z., Zhu, Y., Li, N., and Zhao, D. (2019). A survey of deep reinforcement learning in video games.

[Sherstinsky, 2020] Sherstinsky, A. (2020). Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306.

[Shlens, 2014] Shlens, J. (2014). Notes on kullback-leibler divergence and likelihood.

[Siami-Namini et al., 2019] Siami-Namini, S., Tavakoli, N., and Namin, A. S. (2019). A comparative analysis of forecasting financial time series using arima, lstm, and bilstm.

[Singh et al., 2019] Singh, L., Singh, S., Arora, S., and Borar, S. (2019). One embedding to do them all.

[Sobolev and Vetrov, 2019] Sobolev, A. and Vetrov, D. (2019). Importance weighted hierarchical variational inference.

[Staudemeyer and Morris, 2019] Staudemeyer, R. C. and Morris, E. R. (2019). Understanding lstm – a tutorial into long short-term memory recurrent neural networks.

[Su et al., 2019] Su, S.-Y., Hajimirsadeghi, H., and Mori, G. (2019). Graph generation with variational recurrent neural network.

[Sun and Bischl, 2019] Sun, X. and Bischl, B. (2019). Tutorial and survey on probabilistic graphical model and variational inference in deep reinforcement learning. *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*.

[Taherdangkoo et al., 2013] Taherdangkoo, M., Paziresh, M., Yazdi, M., and Bagheri, M. (2013). An efficient algorithm for function optimization: modified stem cells algorithm. *Open Engineering*, 3(1).

[Tang and Glass, 2018] Tang, H. and Glass, J. (2018). On training recurrent networks with truncated backpropagation through time in speech recognition. *2018 IEEE Spoken Language Technology Workshop (SLT)*.

[Tokui and sato, 2016] Tokui, S. and sato, I. (2016). Reparameterization trick for discrete variables.

[Tolooshams et al., 2019] Tolooshams, B., Dey, S., and Ba, D. (2019). Deep residual autoencoders for expectation maximization-inspired dictionary learning.

[Turkoglu et al., 2019] Turkoglu, M. O., D'Aronco, S., Wegner, J. D., and Schindler, K. (2019). Gating revisited: Deep multi-layer rnns that can be trained.

[van Opheusden et al., 2020] van Opheusden, B., Acerbi, L., and Ma, W. J. (2020). Unbiased and efficient log-likelihood estimation with inverse binomial sampling.

[Vinayak et al., 2019] Vinayak, R. K., Kong, W., Valiant, G., and Kakade, S. M. (2019). Maximum likelihood estimation for learning populations of parameters.

[Wang and Raj, 2015] Wang, H. and Raj, B. (2015). A survey: Time travel in deep learning space: An introduction to deep learning models and how deep learning models evolved from the initial ideas.

[Wang et al., 2020] Wang, S., Pathania, A., and Mitra, T. (2020). Neural network inference on mobile socs. *IEEE Design Test*, page 1–1.

[Weininger, 1988] Weininger, D. (1988). SMILES, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Modeling*, 28(1):31–36.

[Werbos, 1990] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

[Whitley, 1994] Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing*, 4(2).

[Wilson et al., 2017] Wilson, J. T., Moriconi, R., Hutter, F., and Deisenroth, M. P. (2017). The reparameterization trick for acquisition functions.

[Wu et al., 2020] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, page 1–21.

[Xu et al., 2018] Xu, M., Quiroz, M., Kohn, R., and Sisson, S. A. (2018). Variance reduction properties of the reparameterization trick.

[Ying et al., 2018] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, KDD '18, page 974–983, New York, NY, USA. Association for Computing Machinery.

[You et al., 2018] You, J., Liu, B., Ying, R., Pande, V., and Leskovec, J. (2018). Graph convolutional policy network for goal-directed molecular graph generation.

[Zhang, 2019] Zhang, J. (2019). Basic neural units of the brain: Neurons, synapses and action potential.

[Zhou et al., 2018] Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2018). Graph neural networks: A review of methods and applications.

[Zhuang et al., 2019] Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. (2019). A comprehensive survey on transfer learning.