COMPREHENSIVE STUDY OF GENERATIVE METHODS ON DRUG DISCOVERY

by

SIYU XIU

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON
December 2019

To my family, for their endless trust, support, encourage and love.

ACKNOWLEDGEMENTS

There were countless people who helped me during my master studying career, and I would like to use this opportunity to express my appreciation to them.

I would like to thank my supervising professor Dr. Junzhou Huang for his invaluable advice and guidance in my master thesis study. None of the work in this thesis would have been possible without him.

Furthermore, I would like to thank my thesis committee members Dr. Jia Rao, Dr. Dajiang Zhu for their interest and support in my research and for their valuable suggestions regarding this thesis. It has been a honor for me to have each of them serve in my committees.

The research described in this thesis has benefited from my other collaborators besides my advisors. Without them, some of the chapters in this thesis would not have been so polished. My special thanks go to Dr. Zheng Xu, and Sheng Wang. I have been learning a lot from them through the collaborations.

I want to thank all my colleagues from the Scalable Modeling and Imaging and Learning Lab (SMILE), the Computer Science and Engineering Department. It is my pleasure to meet such a nice squad of creative researchers. I am deeply thankful to all of them with whom I spent my time as a graduate student at UTA.

Finally, my special thanks go to my family. I would like to thank my parents for their love and endless support. Without them, it would not have been feasible for me to achieve these accomplishments in my career.

November 11th, 2019

ABSTRACT

COMPREHENSIVE STUDY OF GENERATIVE METHODS ON DRUG DISCOVERY

SIYU XIU, M.S.

The University of Texas at Arlington, 2019

Supervising Professor: Junzhou Huang

Observing the recent success of the deep learning (DL) technology in multiple life-changing application areas, e.g., autonomous driving, image/video search and discovery, natural language processing, etc., many new opportunities have presented themselves. One of the biggest ones lies in applying DL in accelerating the drug discovery, where millions of human lives could potentially be saved. However, applying DL into drug discovery task turns out to be non-trivial. The most successful DL methods take fix-sized tensors/matrices, e.g., images, or sequences of tokens, e.g., sentences with variant numbers of words, as their inputs. However, none of these registers with the inputs of drug discovery, i.e., chemical compounds. Due to the structural nature of the chemical compounds, *graph* data structure is often used to represent the atomic data for the compound. Seen as a great opportunity for improvement, deep learning on graph techniques are being actively studied lately.

In this paper, we survey the newest academic progress in generative deep learning methods on graphs for drug discovery applications. We will focus our study by narrowing down our scope to one of the most important deep learning generative model, namely Variational AutoEncoder (VAE). We start our survey introduction by dating back to the stage when each molecule atom is treated completely separately and their structural information is completely ignored in VAE. This method is quite limited given their structure information is scraped. We hence introduce the baseline method Grammar Variational AutoEncoder (GVAE) where the chemical representation grammar information is encoded in the modeling. One improvement upon the GVAE is by ensuring the syntax validation in the decoder. This method is named Syntax-Directed Variational AutoEncoder (SDVAE). Since then, a couple of variants of these methods have bloomed. One of them is by encoding and decoding the molecules in two steps, one being junction tree macro structure with chemical subcomponents as the minimum unit and the other one being the micro structure with atom as the minimum unit. This method is named Junction Tree Variational AutoEncoder (JTVAE). Finally, we introduce another method named GraphVAE where the pre-defined maximum atom number is enforced in the decoder. Those methods turn out to be effective in avoiding generating invalid molecules. We show the effectiveness of all the methods in extensive experiments. In conclusion, the light of hope has been lit in the drug discovery area with deep learning techniques when a ton of opportunities for growth are still open.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In this chapter, we will introduce multiple drug discovery problem and it current state of academic research. Also, we will present the current representation learning system for drug compounds. Last but not least, we will survey the previous attempts in addressing the drug discovery tasks.

1.1 Drug Discovery

In recent decades, drug discovery has become an essential problem, where millions of human lives could potentially be saved. The current solution is mostly biological-experiment-based which costs billions of dollars and multiple years before a new drug can be successfully invented and safely landed on markets. Observing the recent success of the deep learning (DL) technology in multiple life-changing application areas, e.g., autonomous driving, image/video search and discovery, natural language processing, etc., many new opportunities have presented themselves. New hope has raised in transferring the success of deep learning to drug discovery area. However, applying DL into drug discovery task turns out to be a non-trivial task. The most successful DL methods takes fix-sized tensors/matrices, e.g., images, or sequences of tokens, e.g., sentences with variant numbers of words, as their inputs. However, none of these registers with the inputs of drug discovery, i.e., chemical compounds. Due to the structural nature of the chemical compounds, *graph* data structure is often used to represent the atomic data for the compound which is still quite under-studied in DL area.

1.1.1 Drug Virtual Screening

Drug virtual screening is a computational technique applied in drug discovery to search libraries of small molecules in order to identify those structures which are most likely to bind to a drug target, typically a protein receptor or enzyme. With deep learning, we can potentially identify the most-drug-like chemical compounds in a faster and more accurate direction.

1.1.2 Drug Generative Methods

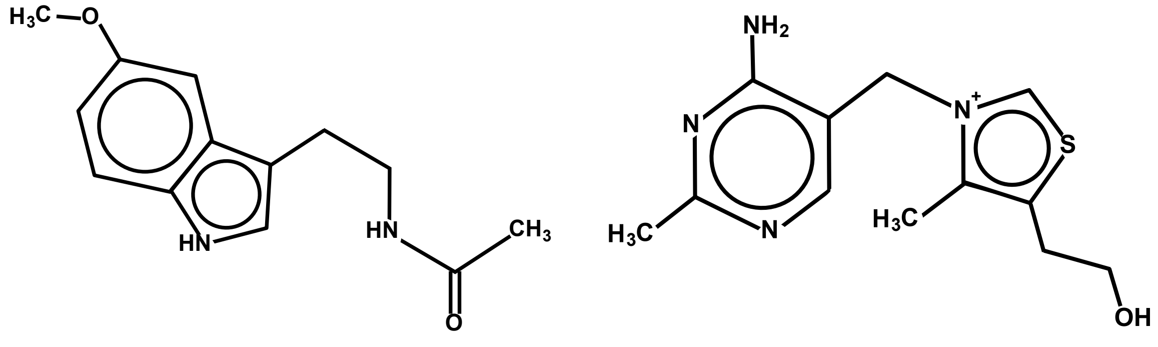
Drug candidate generation is a process of generating the drug candidates directly, often given some chemical property prior by human. The process normally involves biologist identify some task-specific properties for the drug chemical compound. Then the properties are fed into the computational model and the model will automatically generates the drug candidate for further screening. This method is more efficient than virtual screening where the drug search target library is generally as huge as millions of molecules.

1.2 Machine/Deep Learning in Drug Discovery

1.2.1 Digital Drug Representations

1.2.1.1 SMILE Representations of Molecules

Vanilla SMILE Representation System To digitally store the molecules, the molecule needs to be represented in some form of computer data structures. One of most popular representations is a sequence representation named SMILE, which is short for the Simplified Molecular-Input Line-Entry system [5]. SMILE is a line notation for describing the structure of chemical species using text strings. The SMILE system represents the graph-based definition of chemical structures in a text sequence,



(a) (b)

Figure 1.1: The examples of SMILE representations.
a) **Melatonin**: **CC(=O)NCCC1=CNc2c1cc(OC)cc2**
b) **Thiamine**: **OCCc1c(C)[n+](cs1)Cc2cnc(C)nc2N**

where the atoms, bonds and rings are encoded in a graph and represented in text sequences. Simple examples of SMILE representations are 1) hydrogen cyanide with structure $C \equiv N$ (C#N), 2) carbon dioxide with structure $O = C = O$ (O=C=O), where corresponding SMILE representations are included in the round brackets. Here, atoms are represented by the standard symbol of the chemical elements, in square brackets, e.g., [Fe] for iron, etc. However, it might be omitted in some case like when the element is too common, e.g., C for carbon. Bonds are represented using symbols like -, =, for single, double and triple bounds. For more details, readers are referred to [5].

Canonical SMILE: An One-on-one Mapping One problem the vanilla SMILE system has is that the representation method is not a bijective mapping between SMILE sequence and a molecule. For instance, a molecule can correspond to multiple SMILE sequences, e.g., OCC, CCO and $C(O)C$. This is absolutely not ideal in the computer world. For example, the vanilla SMILE representation of molecule is not "hash-able" in the sense that one item can potentially have multiple hash values.

To standardize the representation, we need to provide an one-on-one mapping between SMILE sequences and molecules. Hence, multiple "canonicalization algorithms" are invented to ensure the representation *uniqueness* of each molecular structure [6]. In this paper, all of our SMILEs are canonical SMILEs to ensure the bijectiveness of the mapping [7].

1.2.2 Continuous Representation

One of the major challenge in applying machine/deep learning techniques into drug discovery tasks is the effective continuous representation of the molecular graph. It is the continuous representation that can be thereafter fed into a machine learning system as an initial vector representation. A large number of previous research progress are made to build new continuous representation systems.

1.2.2.1 Hand-crafted Fingerprint Methods

Previously, there is a major class of molecular representation system called **fingerprint**, which is essentially a vector representation of a corresponding molecule. Those methods are usually manually crafted. Among those hand-crafted methods, there are hash-based methods and biological-property-guided fingerprint methods.

There has been many hash-based methods developed to generate unique molecular feature representation [8, 9, 10]. The most fundamental and popular one is called **circular fingerprints**. Circular fingerprints generate each layer's features by applying a fixed hash function to the concatenated features of the neighborhood in the previous layer. Based on the hash method used, the method can be further categorized, with Extended-Connectivity Finger-Print (ECFP) [11] being one instance of those methods. However, due to the non-invertibility of hash functions, the hash-bashed fingerprint methods usually do not embed enough information in the continuous representation. This usually leads to insufficient performance in the subsequent discovery tasks.

On the other hand, there are other fingerprint methods that are designed based on the biological experiments and the expertise knowledge and experience, e.g., [12, 13]. First, the task should be determined before generating the representation. Then, biologists search for several most task-relevant sub-structures (fragments), e.g., $CC(OH)CC$ for pro-solubility prediction, and count those sub-structures as local features to produce fingerprints. This kind of fingerprint methods usually work well for specific tasks, but poorly generalize for other tasks.

1.2.2.2 Graph Neural Network Fingerprint Methods

So far, deep learning methods have demonstrated their power on supervised learning problems. Following this trend, many of deep learning-based fingerprint methods are still trained in a supervised-learning fashion [14, 15]. For these methods, they only use labeled molecular data samples. These methods take vanilla graph embedding as inputs [16, 17, 18, 19] and update model weights via back-propagation based on the loss computation between prediction and ground truths. However, as mentioned earlier, the performance of the deep supervised learning models are particularly limited by the quality and quantity of the labeled data. Among them, the state-of-the-art work is the neural fingerprint [20]. By mimicking the process of generating circular fingerprint, the neural fingerprint method generate each layer's features by applying a fixed hash function to the concatenated features of the neighborhood in the previous layer. However, instead of using a fixed hash function, it utilizes a non-linear activated densely connected layer.

1.2.2.3 Seq2seq Fingerprint

One interesting idea of generating fingerprint is by using sequence to sequence (seq2seq) model ,which is originally used in natural language translation, e.g. English-to-French text translation, to generate molecular embedding. The high-level idea of seq2seq fingerprint [21] is to treat the molecular SMILE sequence as a "sentence" and translate the it back to itself. The intermediate embedding will encode all the information needed to recover the SMILE sequence. The seq2seq fingerprint/embedding can be pipe-lined to other supervised/semi-supervised training with other models, e.g., Adaboost [22], GradientBoost [23], and RandomForest [24], etc. Since the seq2seq fingerprint model training does not require any ground truths, the representation is expected to embed sufficient information to recover itself and provide enough inference power. Given it is usually trained with a huge pool of unlabeled valid SMILE sequences. It is robust to the specific labeled task, but might not provide optimal inference performance for each task.

1.2.3 Generative Methods

1.2.3.1 Variational Auto-Encoder

Variational Auto-Encoder (VAE) model [25] is a variant type of artificial neural network used to learn efficient data codings in an unsupervised manner, which uses a *encoder* to encode the original representation to a vector or scalar then a *decoder* to decode the vector to original representation. The difference is that the VAE model puts the assumption that the embedded space follows some specific Gaussian distribution. It becomes increasingly popular to use VAE to generate (especially smaller) molecular graphs.

1.2.3.2 Generative Adversarial Network

Generative Adversarial Network (GAN) [26] has recently become popular in the machine learning area. A GAN is constructed by a *discriminator* and a *generator*. The discriminator acts as a cop to distinguish the training data samples from the samples generated from the generator. Hence, the learning process actually learns from both training data set and the generated fake data samples. It works well when the scale of data sample is limited. But such network is harder to be trained to converge to a reasonable solution.

1.3 Thesis Overview

So far, we have introduced the multiple tasks in drug discovery, the representation system of molecules and previous most relevant attempts in addressing multiple drug discovery tasks. In this section, we will briefly overview the structure of the entire thesis. In Chapter 2, we introduce the baseline method of variational auto-encoder on molecule small graphs, namely Grammar Variational AutoEncoder (GVAE). In Chapter 3, one improvement over GVAE is presented as the syntax validation is induced in the decoder of the VAE. This method is named as Syntax-Directed Variational AutoEncoder (SDVAE). In Chapter 4, we introduce the junction tree-based variational auto-encoder for molecular graph generation. Then, Chapter 5 introduces the GraphVAE methods where the human prior of pre-defined maximum graph vertex number is explicitly encoded in the graph. In Chapter 6, we present the experimental results in comparing above methods.

As the ending, Chapter 7 draws our conclusions of the thesis, where we summarize the presented deep learning [27, 28] methods for molecular graph generation and provide some future research directions.

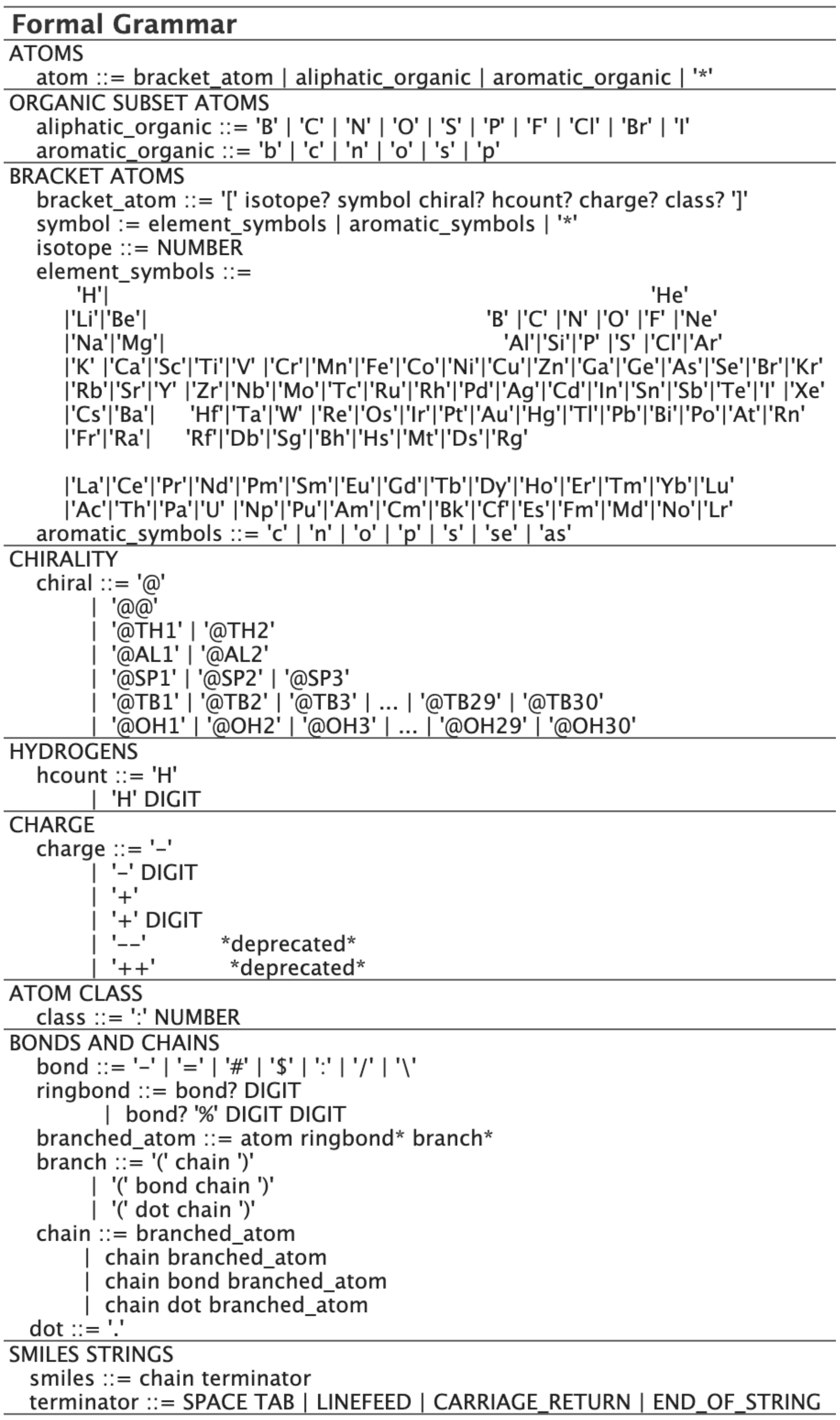
CHAPTER 2

GRAMMAR VARIATIONAL AUTOENCODER

In this chapter, we introduce the Grammar Variational AutoEncoder method [1]. Traditionally, a context-free grammar (CFG) consists of a finite set of non-terminal symbols (V), a finite set of terminal symbols (Σ), a finite set of production rules (R), and a different non-terminal symbol (S). It can be expressed as $G = (V, \Sigma, R, S)$ where Σ does not intersect V. In fact, production rules R can be considered as a rule rewrite. $\alpha \to \beta$ for $\alpha \in V$ and $\beta \in (V \cup \Sigma)*$, where $*$ is a unary operation, either on sets of strings or on sets of symbols or characters, is the formal formula to present the rules R. The formal grammar is shown in Figure 2.1.

2.1 Methods

If we assume right-hand-side symbols of the production rule is child nodes of left-hand-side symbols, the process of apply production rules becomes the process of building a tree. Thus, grammar G can be seem as a series of production rule, parent-child correspondences. After repeatedly calling the corresponding R rule, all leaf nodes are terminal symbols in . From left to right of the leaf nodes, set of all sequences of the leaf nodes is known as the language of G. Figure 2.2 shows how the process of apply production rules becomes the process of building a tree. A parse tree is the tree which S is the root and its leaf nodes is one of the language. For example, if "AGET" is a string in the language, then the parse tree is the tree which has S as its root and the sequence of leaf is "AGET".



```
Formal Grammar
ATOMS
   atom ::= bracket_atom | aliphatic_organic | aromatic_organic | '*'
ORGANIC SUBSET ATOMS
   aliphatic_organic ::= 'B' | 'C' | 'N' | 'O' | 'S' | 'P' | 'F' | 'Cl' | 'Br' | 'I'
   aromatic_organic ::= 'b' | 'c' | 'n' | 'o' | 's' | 'p'
BRACKET ATOMS
   bracket_atom ::= '[' isotope? symbol chiral? hcount? charge? class? ']'
   symbol := element_symbols | aromatic_symbols | '*'
   isotope ::= NUMBER
   element_symbols ::=
        'H'|                                                            'He'
      |'Li'|'Be'|                                      'B' |'C' |'N' |'O' |'F' |'Ne'
      |'Na'|'Mg'|                                      'Al'|'Si'|'P' |'S' |'Cl'|'Ar'
      |'K' |'Ca'|'Sc'|'Ti'|'V' |'Cr'|'Mn'|'Fe'|'Co'|'Ni'|'Cu'|'Zn'|'Ga'|'Ge'|'As'|'Se'|'Br'|'Kr'
      |'Rb'|'Sr'|'Y' |'Zr'|'Nb'|'Mo'|'Tc'|'Ru'|'Rh'|'Pd'|'Ag'|'Cd'|'In'|'Sn'|'Sb'|'Te'|'I' |'Xe'
      |'Cs'|'Ba'|     'Hf'|'Ta'|'W' |'Re'|'Os'|'Ir'|'Pt'|'Au'|'Hg'|'Tl'|'Pb'|'Bi'|'Po'|'At'|'Rn'
      |'Fr'|'Ra'|     'Rf'|'Db'|'Sg'|'Bh'|'Hs'|'Mt'|'Ds'|'Rg'

      |'La'|'Ce'|'Pr'|'Nd'|'Pm'|'Sm'|'Eu'|'Gd'|'Tb'|'Dy'|'Ho'|'Er'|'Tm'|'Yb'|'Lu'
      |'Ac'|'Th'|'Pa'|'U' |'Np'|'Pu'|'Am'|'Cm'|'Bk'|'Cf'|'Es'|'Fm'|'Md'|'No'|'Lr'
   aromatic_symbols ::= 'c' | 'n' | 'o' | 'p' | 's' | 'se' | 'as'
CHIRALITY
   chiral ::= '@'
          |  '@@'
          |  '@TH1' | '@TH2'
          |  '@AL1' | '@AL2'
          |  '@SP1' | '@SP2' | '@SP3'
          |  '@TB1' | '@TB2' | '@TB3' | ... | '@TB29' | '@TB30'
          |  '@OH1' | '@OH2' | '@OH3' | ... | '@OH29' | '@OH30'
HYDROGENS
   hcount ::= 'H'
          |  'H' DIGIT
CHARGE
   charge ::= '–'
          |  '–' DIGIT
          |  '+'
          |  '+' DIGIT
          |  '––'        *deprecated*
          |  '++'        *deprecated*
ATOM CLASS
   class ::= ':' NUMBER
BONDS AND CHAINS
   bond ::= '–' | '=' | '#' | '$' | ':' | '/' | '\'
   ringbond ::= bond? DIGIT
            |  bond? '%' DIGIT DIGIT
   branched_atom ::= atom ringbond* branch*
   branch ::= '(' chain ')'
          |  '(' bond chain ')'
          |  '(' dot chain ')'
   chain ::= branched_atom
         |  chain branched_atom
         |  chain bond branched_atom
         |  chain dot branched_atom
  dot ::= '.'
SMILES STRINGS
  smiles ::= chain terminator
  terminator ::= SPACE TAB | LINEFEED | CARRIAGE_RETURN | END_OF_STRING
```

Figure 2.1: This is the Formal Grammar. In each box, there are several production rules (R). Also, as shown, it contains terminal symbols (Σ) in the SMILES STRINGS part and non-terminal symbol (S) in ORGANIC SUBBSET ATOMS, BRACKET ATOMS, CHIRALITY, HYDROGENS, CHARGE, and BOND AND CHAINS.[1]

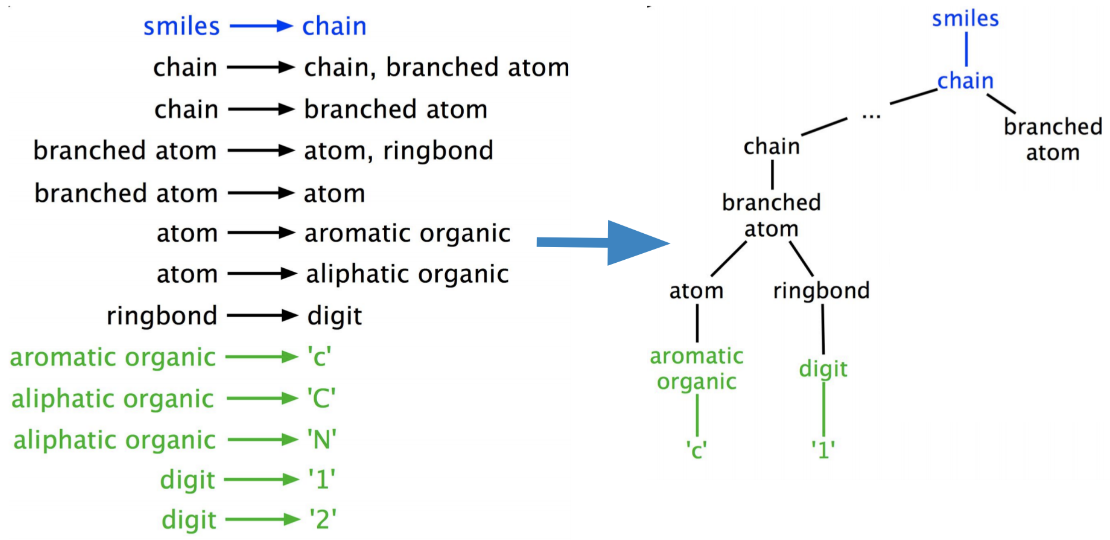


Figure 2.2: The process of apply production rules becomes the process of building a tree. The right-hand-side symbols of the production rule is child nodes of left-hand-side symbols. The right side of this figure is the parse tree transform from production rules. [1]

The mainstay of the probability generation model of a valid string can be composed by a context-free grammar.By assigning probabilities to each production rule in the grammar, a probability distribution on the parse tree can be defined [29, 30].The way to generate a string is to iteratively sample from the start symbol and implement the production rules if there are non-terminal symbols.The probability that modern methods allow for use at each stage depends on the current state of the parse tree [31].

As known, character variational autoencoders has a disadvantage that it may frequently map latent points to invalid sequences instead of valid sequences. According to the grammar, We can parse any valid sequence into a sequence of production rules which applied in order will product the original sequence. The process that applying sequence of production rules in order to product the original sequence is shown in Figure 2.3 Also, the valid set of rules can be selected at any time in generating. Therefore, grammar variational autoencoders can commit to learning semantic properties of sequence data instead of learning syntactic constraints at same time. there is a specific example which can hope to understand the grammar variational autoencoder.

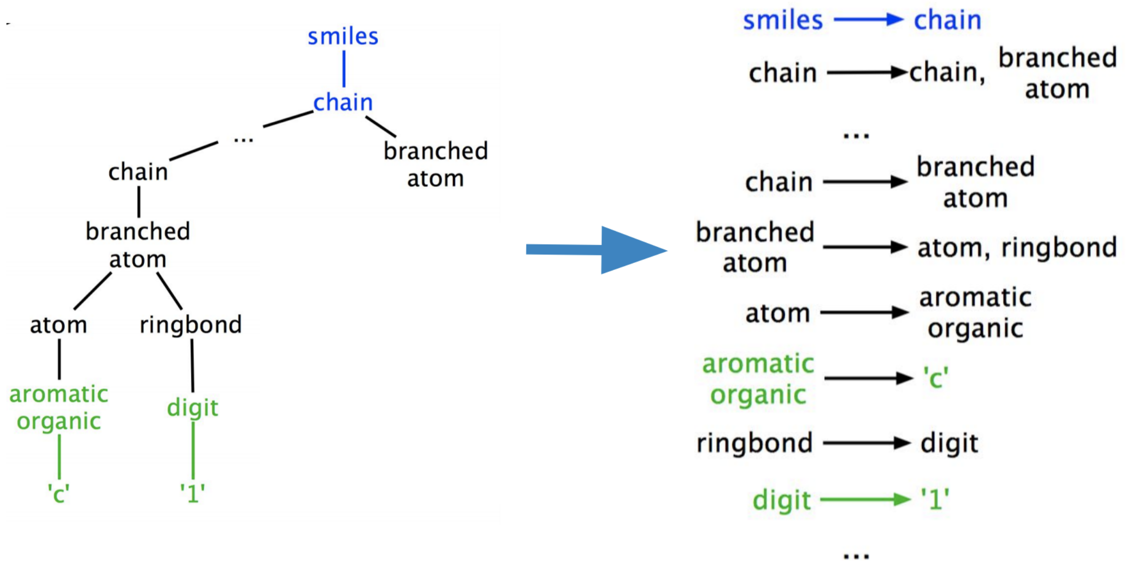


Figure 2.3: Applying a sequence of production rules in order will product the original sequence. The left side of this figure is the parse tree, sequence of production rules. The root is SMILES which is the left part of the first rule. CHAIN is its child which is on the right side of first rule. Therefore, it is obvious that the parent node in the parse tree is the right-hand-side symbols of the production rule, and their children are located in the left-hand-side. [1]

2.2 Encoding

2.2.1 Context-free Grammars

Normally, to analyze the Grammar VAE, we separate it into two parts, encoding and decoding. The whole process of encoding is shown in Figure 2.4. In Figure 2.5, there is a subset of the SMILES grammar on the left side which includes many

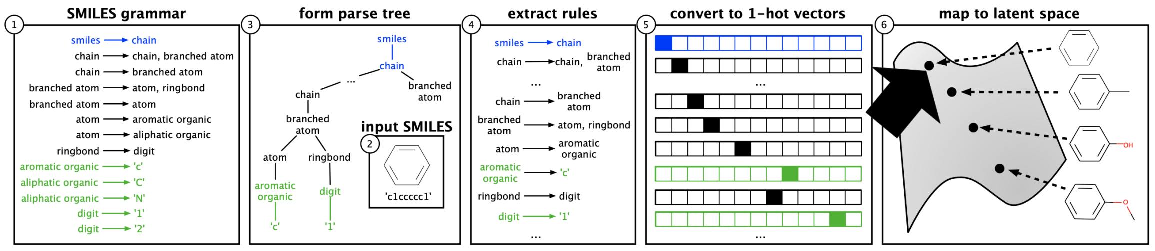


Figure 2.4: This is the process of encoding. It includes six boxes. It contains the process that how SMILES grammar is transformed to parse tree, hot to extract rules from parse tree, how convert rules to 1-hot vectors, and how to achieve information from vectors and map it to latent space.[1]

production rules that may be used for constructing a molecule. Assume that the input SMILES is ‘c1ccccc1’ which has the molecule shown on the right of Figure 2.5.

First, we transform this string into a parse tree, shown on the right of Figure 2.5 , by utilizing the SMILES grammar. After this process, molecule is encoded to a continuous latent portrayal.

Then, we break up this tree into several production rules. To implement this splitting, we apply the pre-order depth-first search. Every formula which present the relation between parent and child is seemed as a production rule. Thus, after traversing this tree, we achieve a sequence of production rules shown on the right side of Figure 2.6.

Next, these rules are transformed into 1-hot indicator vectors. Each dimension in the 1-hot indicator vectors matches a rule in the SMILES grammar. This process is shown as Figure 2.7. The size of the 1-hot indicator vectors is $T(X) \times K$ matrix X, where K is the total number of production rules of the entire grammar and $T(X)$ is the total number of productions rules which is utilized in whole encoding process.

In the last step of encoding, as shown in 2.8, we map the 1-hot indicator vectors to a continuous latent vector by using a regularized versions of multi-layer perceptrons, CNN.

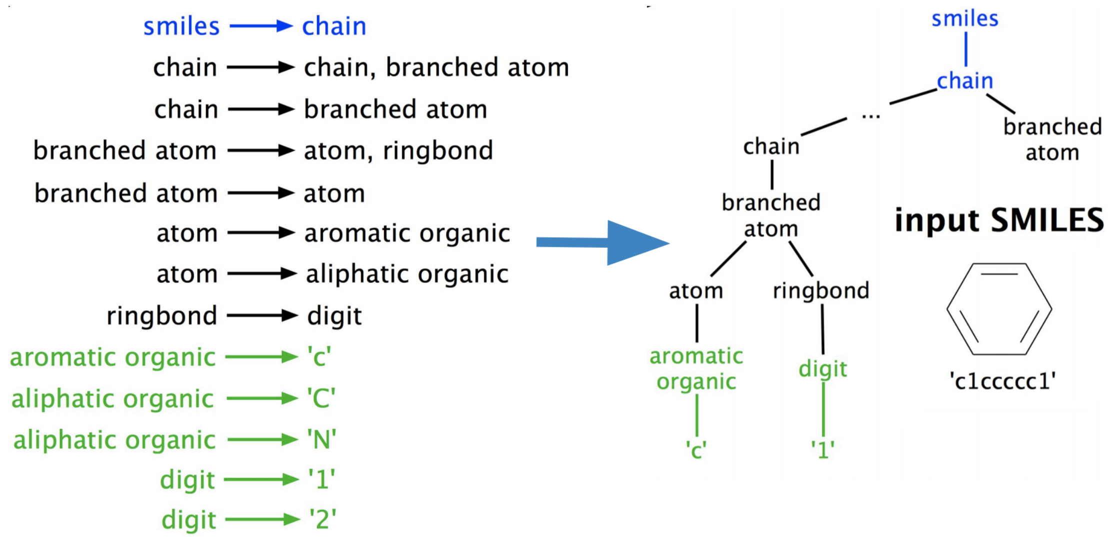


Figure 2.5: This is the process how to transform SMILES grammar to parse Tree with the specific input SMILES. As we can see the first rule is start from SMILES symbol. Thus, SMILES is the root of parse tree. The left-hand-side symbols of the production rule is parent nodes of right-hand-side symbols. After the whole process, from left to right of the leaf nodes marked in green, set of all sequences of the leaf nodes is as same as the original SMILES, 'c1ccccc1'. [1]

2.3 Decoding.

After mapping a sequence of production rules to continuous latent vector, we need to map them back. In this process, we create a decoder. The whole process of decoding is shown in Figure 2.9. This decoder can help to mask out invalid rules and only choose valid rules. Therefore, only the valid parse sequence can be produced. At the beginning, we extract the continuous vector from latent space and convert it to a set of unnormalized log probability vectors (or 'logits'), shown in Figure 2.10. As same as the 1-hot vectors in encoder, each dimension in the logit vectors matches a rule in the SMILES grammar. Thus, matrix can be used to present these collection of logit vectors as well. Tmax is the maximum number of production rules allowed by the decoder, so the matrix can be shown as $F \in \mathbb{R}^{T_{max} \times K}$.

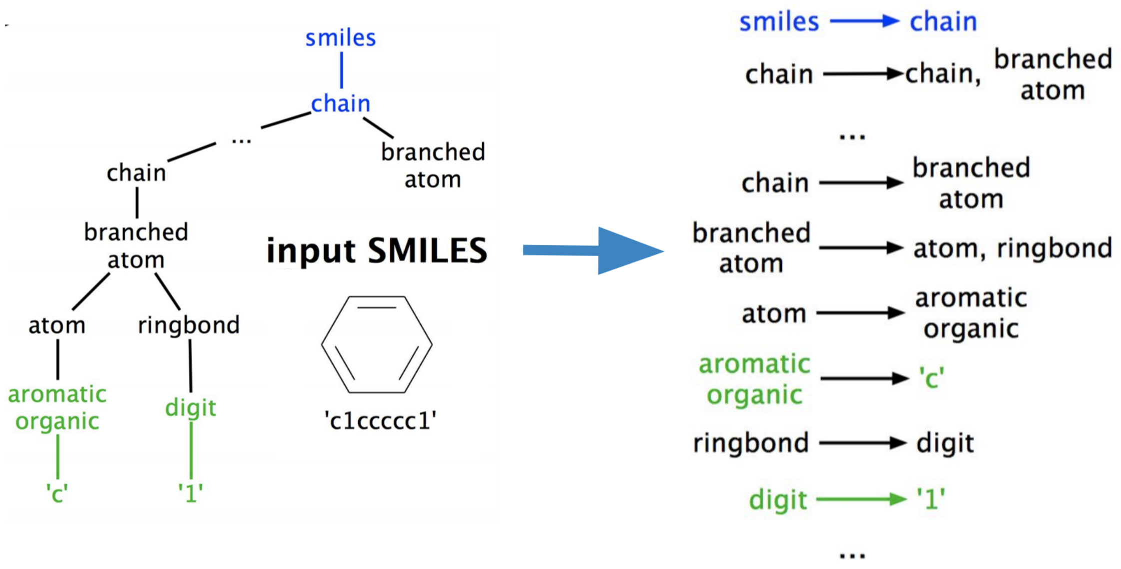


Figure 2.6: This describe how apply a sequence of production rules which is transformed from real SMILES to product the original sequence. Because the root of parse tree is SMILES, the first sequence is the rule start from SMILES and CHAIN is its child. As same as the process of creating parse tree, the parent node in the parse tree is the right-hand-side symbols of the production rule, and their children are located in the left-hand-side. The green symbols are the leaf nodes[1]

Next, we keep tracking the status of the anatomising through a last-in first-out (LIFO) stack to guarantee that we only achieve valid production rules from the decoder. It is significant that smiles on the stack must be the start symbol of every valid parse. Then, we pop off the next non-terminal symbol on the top of the stack. In addition,it is utilized to mask out the invalid dimensions in the logit vector. Officially, we define $m_\alpha \in [0, 1]^K$ as a fixed binary mask vector where is non-terminal. In the other words, at the beginning of process, the first and only production rule in the grammar begin with smiles. So, we only keep the first dimensions and zero-out the rest, shown in Figure 2.11. We then utilize values of the rest of unmasked rules in

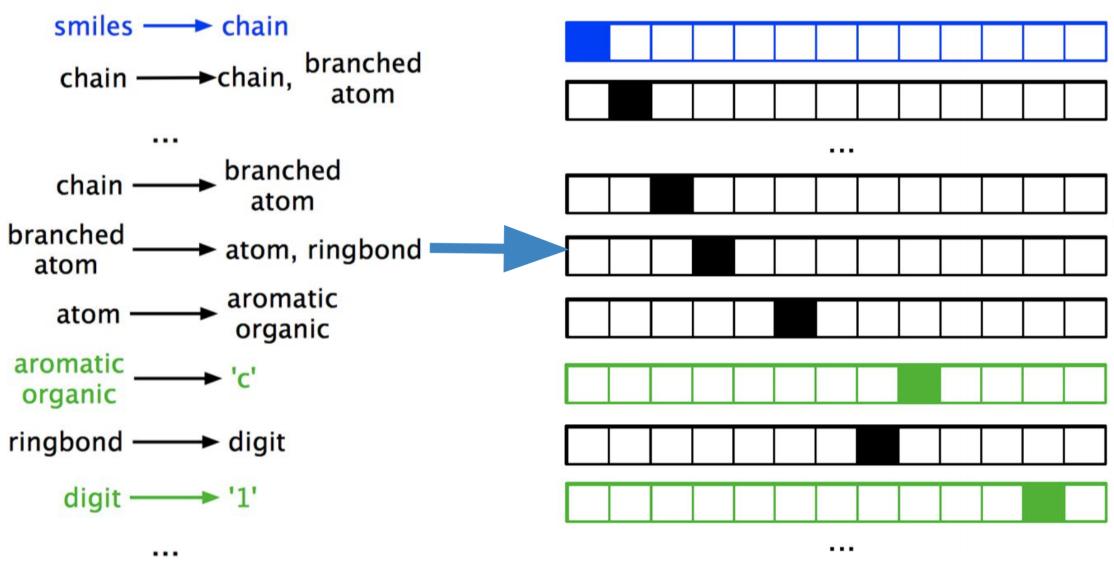


Figure 2.7: Convert rules to one hot vectors. Each dimension in the 1-hot indicator vectors matches a rule in the SMILES grammar. When the rule translate from parse tree can match with one dimension, the dimension will be marked. After apply this method iteratively, all used rule could be marked in one hot vectors. That is how convert rules to one hot encoding. [1]

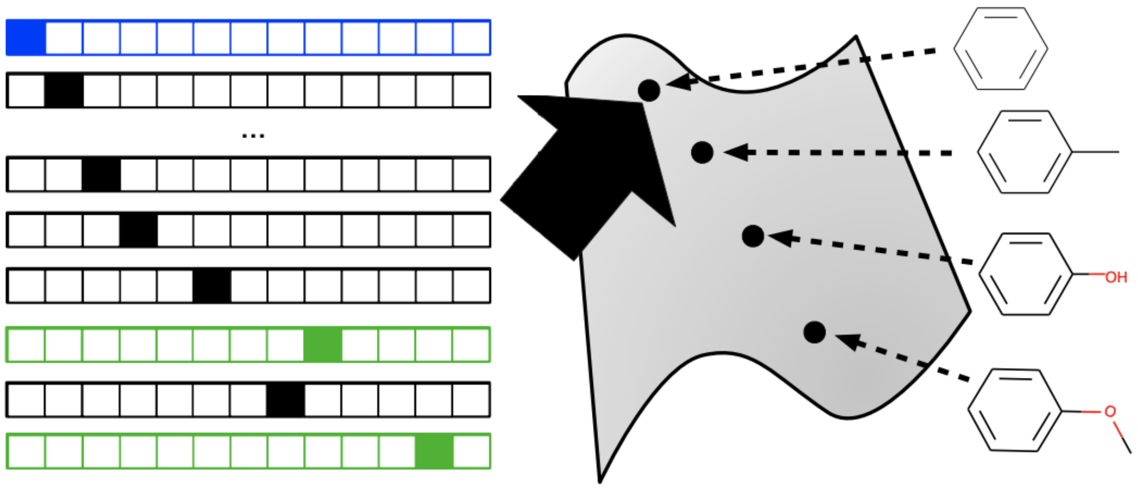


Figure 2.8: Map one hot vectors to latent space. [1]

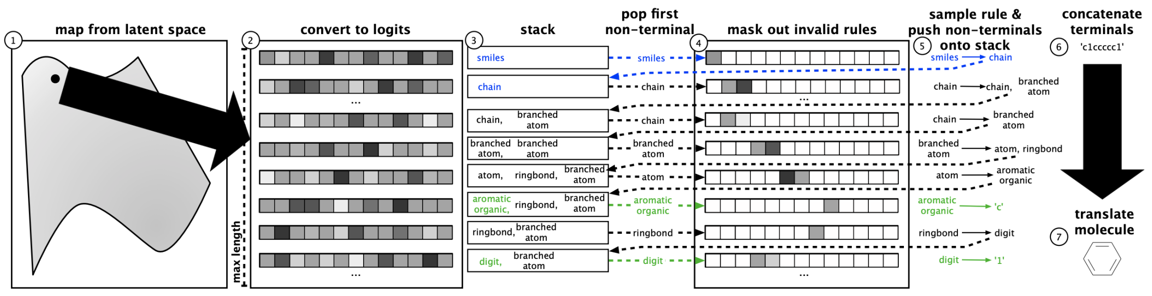


Figure 2.9: This is the process of decoding. Similarly, it includes six boxes which describe how extract information from latent space and convert it to vectors, how to utilize stack to mask out invalid rules from vectors, and how use the after marked vector to achieve objective SMILES and translate it to molecule.[1]

the logit vector to sample. To sample from this masked logit at any timestep t we form the following masked distribution:

$$p((x)_t = k|\alpha, \mathbf{z}) = \frac{m_{\alpha,k}\exp(f_{tk})}{\sum_{j=1}^{k} m_{\alpha,k}\exp(f_{tj})}, \tag{2.1}$$

where f_{tk} is the (t,k)-element of the logit matrix $\mathbb{F}$.Because all rules are marked expect the first rule, this rule smiles → chain will be selected as the first rule in our generated sequence.

Then, we push next rule onto the stack,sample and mask out all of invalid rules. As show in Figure 2.11, we achieve the vector, chain chain, branched atom. Because there are more than one elements on the right side of the vector, we push the nonterminals into stack from right to left. Thus, the element on the top of the stack is the leftmost nonterminal in the vector. Then, we pop the last rule on top of the stack, mask out invalid rules, and push non-terminals onto stack iteratively until the stack is empty or the number of logit vectors is maxium. The pseudocode in Algorithm 1 provides the detail and technological process of this iteration.

Obviously, because of processes mask out invalid rules and push non-terminals onto stack, GVAE always select syntactically-valid sequences. Oppositely, character

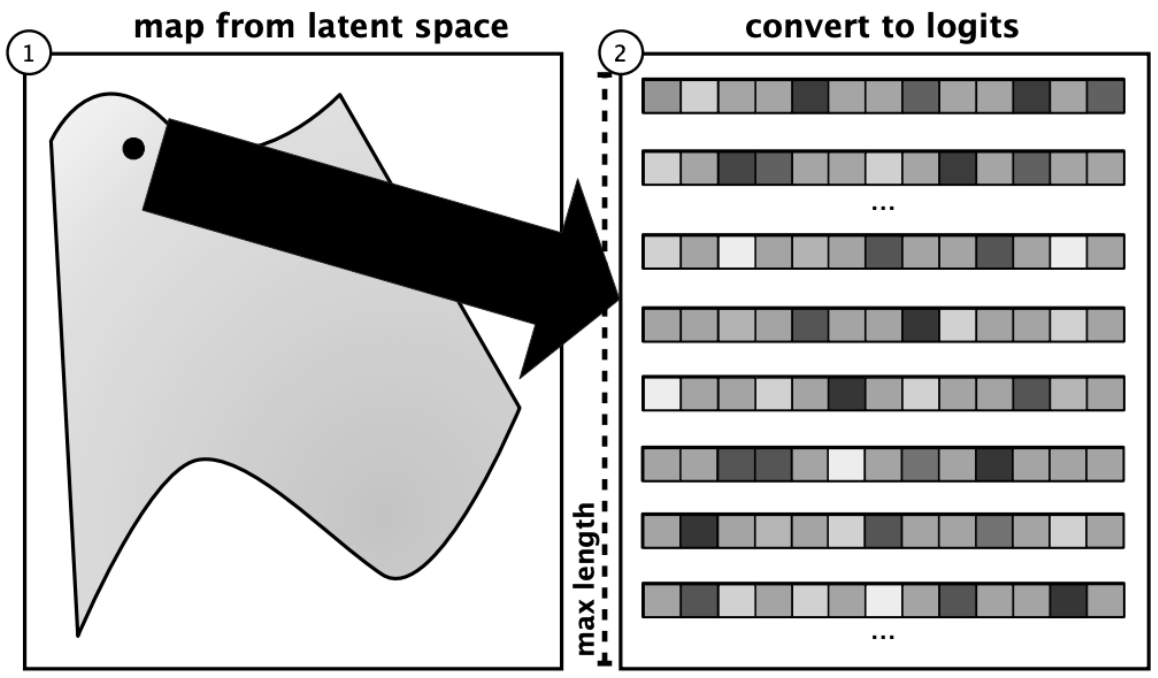


Figure 2.10: The process of mapping continuous vectors from latent space and convert them to logits is shown in the figure. Because continuous vectors are translate by taking the logarithm, the dimension has different colors. Also, due to the difference in log values, it is easy to mask out invalid rules. [1]

VAE sample any possible character due to no stack or masking operation. However, syntactically valid molecules doesn't mean semantically valid molecules. One possible reason is using grammar may create unstable molecules or not chemically-valid which means not exist in real life. Another reason is the the non-context free feature of SMILES. Another reason is the non-context free portion of SMILES. because digits are not nested, it is harder to match digits than match grouping symbols. In addition, for each ringbond, the process of tracking digits is not context-free. Also, it is possible that the GVAE can output an indeterminate sequence if stack still have non-terminal symbols after achieve the T_{max}. There are two way to solve this problem. One is converting these non-terminals to the terminal. The other one, which we utilize, is marking these sequences as invalid.

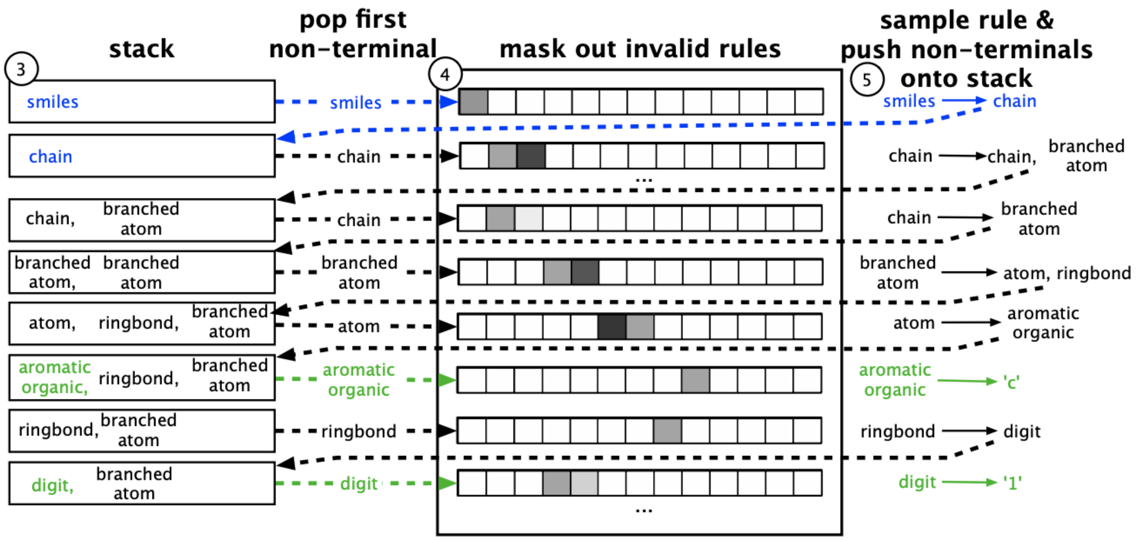


Figure 2.11: This is the whole process of mask out invalid rules. By using last-in first-out stack, only valid production rules can be achieved. First, we push SMILES symbol into stack. Because it is the start label, we only keep the first production rule in vectors and remove zero-out dimension from the second to the rest. By apply the rule we just achieved, we can get the next symbol, CHAIN. We push it into the stack. Then, we utilize values of the rest of unmasked rules in the logit vector to sample. After sample other symbols can be achieved, and we push them into last-in first-out stack. By applying this process iteratively, several letter and digit can be extracted which are factors of SMILES from left to right. [1]

CHAPTER 3

SYNTAX-DIRECTED VARIATIONAL AUTOENCODER

In the previous chapter, we introduce the Grammar Variational AutoEncoder (GVAE) [1], which encodes the grammar information into the variational autoencoder (VAE) to generate syntax valid molecules. However, Grammar Variational AutoEncoder is still incapable to regularize the model for generating semantically valid molecules. For example, in general, the rings in the molecule should be closed to be semantically reasonable. In this case, the Grammar Variational AutoEncoder cannot guarantee the semantic validity. That being said, more constrains need to be introduced into the variational autoencoder modeling.

In this chapter, we present the Syntax-Directed Variational AutoEncoder (SD-VAE) [2] which is another improvement upon the basic character variational autoencoder (CVAE). Syntax-direct variational autoencoder borrows the idea from compiler theory by attaching semantics to a parse tree generated by a context-free grammar (CFG). The syntax-directed generative mechanism in the decoder is able to further constrain the output space hence to ensure the semantic correctness in the molecule generation process.

3.1 Attribute Grammar for SMILES

Attribute grammar is a formal way to define attributes for the productions of a formal grammar, associating these attributes with values. There are two groups of attributes: *synthesized attributes* and *inherited attributes.* A synthesized attribute is computed from the values of attributes of the children. An inherited attribute at a node in parse tree is defined using the attribute values at the parent or siblings.

Ringbond matching Ringbond is in general σ-bond. When representing in graphs, it comes in pairs of a single bond plus a double bond. Each pair should be associated with an index and a bond-type. It forms as an attribute grammar named as ringbond matching. This attributes grammar also generates to other languages, even some computer programming languages where the parentheses of brackets should match.

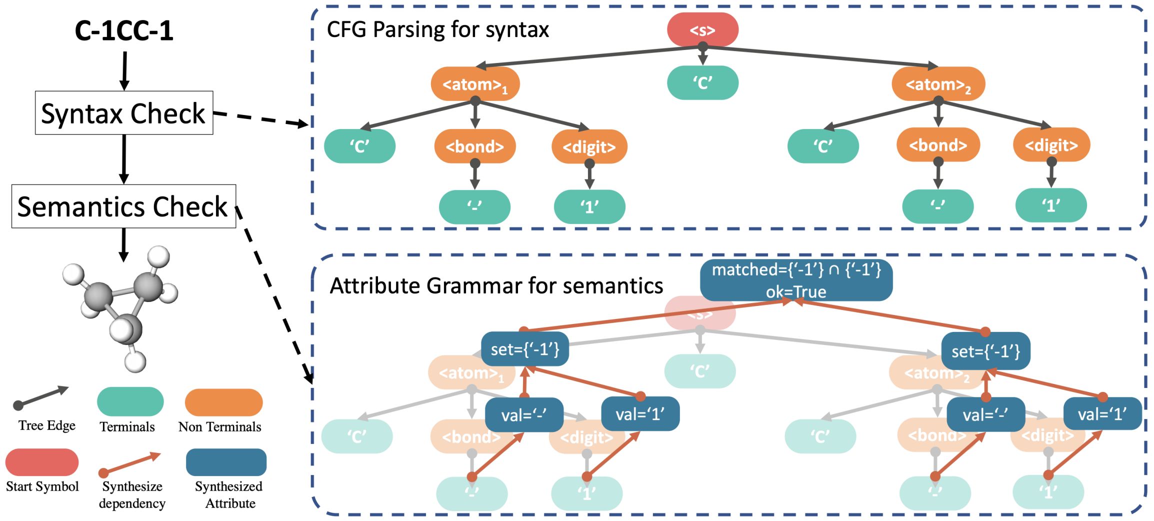


Figure 3.1: In the top, he example of context-free grammar parsing is presented in syntax-directed variational autoencoder. The bottom is the cross-serial dependencies check as semantic check for SMILE string [2].

Explicit valence control In chemistry, a valence electron is an outer shell electron that orbits around an atom, and that usually takes part in the formation of a chemical bond especially when the outer shell is not closed. In a single co-valent bond, both atoms in the bond contribute one valence electron in order to form a shared pair. However, the valence electrons cannot form the bond wildly. There are certain restrictions on the number of bonds each type of atom can form. For example, oxygen has six electrons in its outer shell, needs two more, and will form two covalent bonds to get those two additional electrons. Thus, oxygen is said to have a valence of two. Similarly, carbon has four electrons in its outer shell, so it have a valence of four. Those examples constructs the upper limits of the valance which is also an important constrain in SMILE semantic structure.

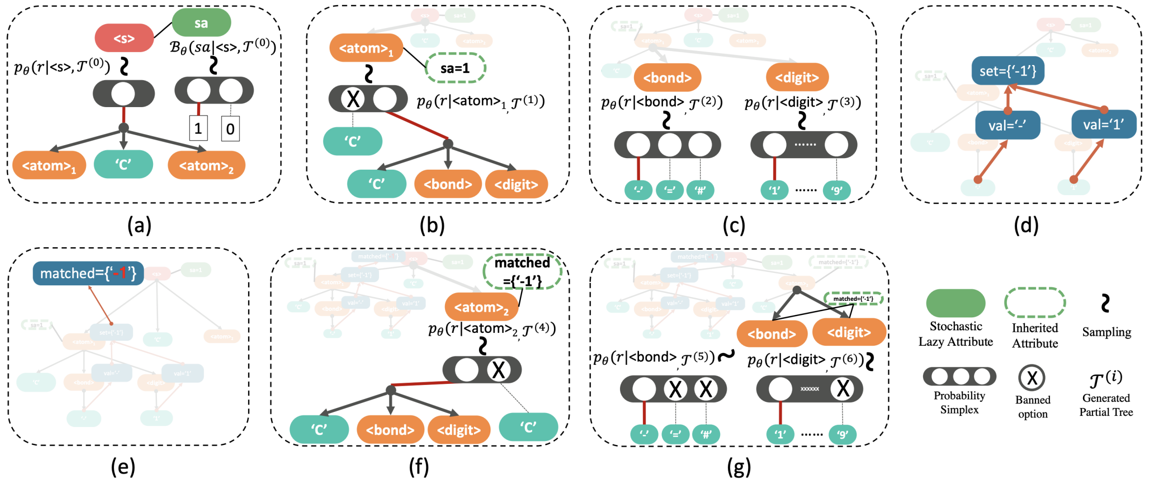


Figure 3.2: The illustration example of how the decoder sample the stochastic lazy attributes for semantic check [2].

In this chapter, we have presented the Syntax-Directed Variational AutoEncoder (SDVAE) method for molecule graph generation with application in drug discovery. This approach introduces additional structural constrain on the decoder part of the variational autoencoder model. In addition to the grammar information used in Grammar Variational AutoEncoder (GVAE), the Syntax-Directed Variational AutoEncoder (SDVAE) incorporate the attribute grammar as an decoder constrain to produce the semantically reasonable molecule generation. This addresses the incapability of semantic validity regularization of decoder, which drastically improves the validity and reconstruction accraucy of the variational autoencoder.

CHAPTER 4

JUNCTION TREE VARIATIONAL AUTO-ENCODER FOR MOLECULAR GRAPH GENERATION

In previous chapters, we introduces the baseline method, namely Grammar Variational AutoEncoder (GVAE), which uses the Grammar information in the variational autoencoder and improves the validity and reconstruction accuracy in drug discovery applications. Also, we have introduces another improvement brought by enforcing syntax constrain in the decoder part of the Variational Autoencoder (VAE). This method is called Syntax-Directed Variational AutoEncoder (SDVAE).

In this chapter, we bring up yet another variant called Junction Tree Variational AutoEncoder (JTVAE) [3]. This method uses two-step fashion. The first step generates a junction tree-structured scaffold over chemical substructures and then combines multiple scaffolds into molecules using the Syntax-Directed Variational AutoEncoder (SDVAE).This method significantly improves the validity of the molecule generation.

Junction Tree The junction tree algorithm, aka, Clique Tree, is a method used in machine learning to extract a subset of a collection of random variables is the probability distribution of the variables contained in the subset in general graphs. In short, it entails performing a breadth search on a modified graph called a **junction tree**. The graph is called a tree because it connected, acyclic and rooted.

For molecule graphs, they are naturally connected. However, it is tricky remove cycles from molecule graphs. In this paper, the basic crux to eliminate cycles is by clustering them into single nodes. It is natural in chemical world given there are multiple rings with chemical priors. There are different algorithms to meet specific needs and for what needs to be calculated. In this paper [3], tree decomposition method [32] is used to merge multiple rings into a single node. Hence the molecule graph can form a junction tree after the extra step.

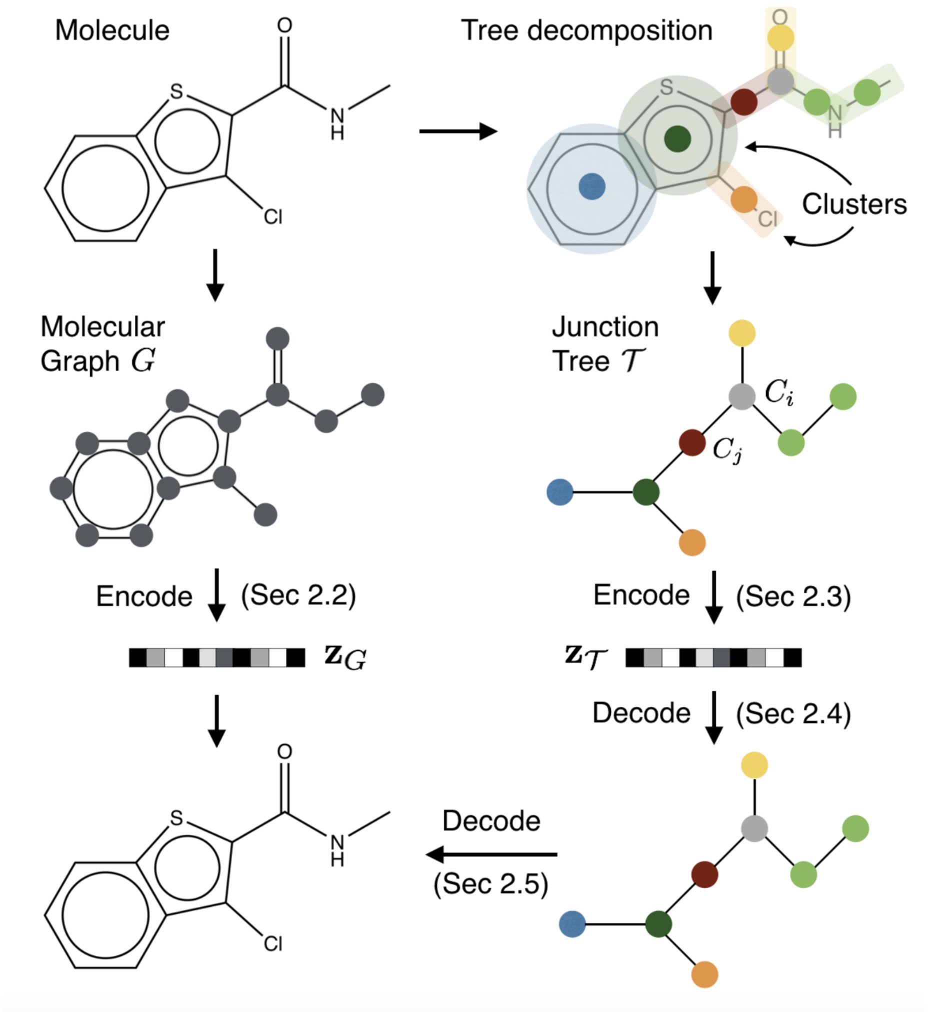


Figure 4.1: The overall process of junction tree variational autoencoder (JTVAE). [3]

In figure 4.2, we show an example process of the junction tree decomposition process. The basic idea is to separate atom clusters and group it as a single node in the tree. A simple ring and branch atom is easier to separate out. The difficulty here is when multiple ring share a same set (two or more) of atoms. If that case happens, in Junction Tree Variational AutoEncoder (JTVAE) method, they will be merged into one cluster as bridged rings.

In the bottom of the figure 4.2, we also show an example set of chemical substructure vocabularies. One can read rings, branch atoms, etc., in the chemical substructure cluster vocabulary.

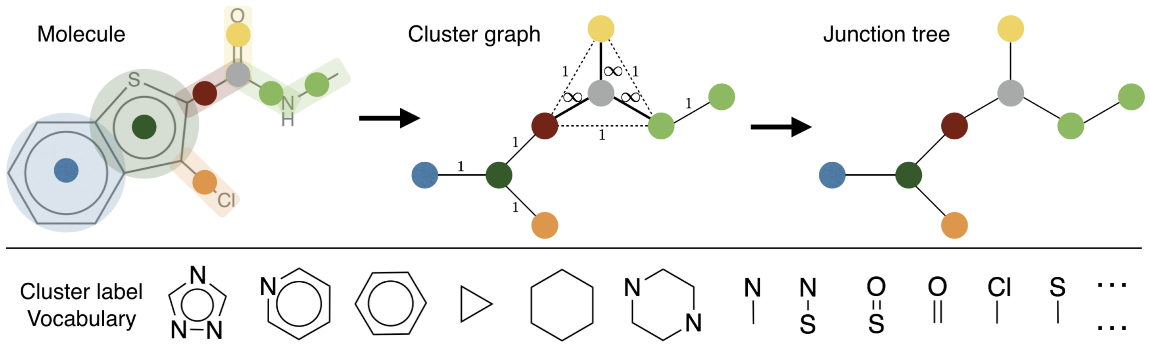


Figure 4.2: When multiple rings share atoms, as shown in the light blue and light green circles, they will be merged as a single cluster (bridged rings).

To conclude this chapter, we introduced the junction tree variational autoencoder (JTVAE) method. The method generates the molecules in a two-phase approach. The first step will generate the molecule scaffold which is the bird-view structure of the molecule. As a second step, the finegrained detail of the molecule is generated. The two step approach ensures the overall validity of the entire molecule graph and hence avoid the invalid generation of molecules in practice.

CHAPTER 5

GRAPHVAE: TOWARDS GENERATION OF SMALL GRAPHS USING VARIATIONAL AUTOENCODERS

In this chapter, we discuss a largely different method from previous three ones called GraphVAE which is short for Graph Variational AutoEncoder. Instead of encoding a sequence of grammar rules as previous three methods do, this method directly encodes the graph into adjacency matrices, edge attribute matrices and node attribute matrices and form as model input. Compared with pure Character Variational AutoEncoder (CVAE), it preserves a lot of structure information and hence provides better reconstruction quality. Also, the variational auto-encoder's decoder is co-regularized by pre-defined maximum graph size. This method shows an drastic improvement, especially on smaller molecule graphs.

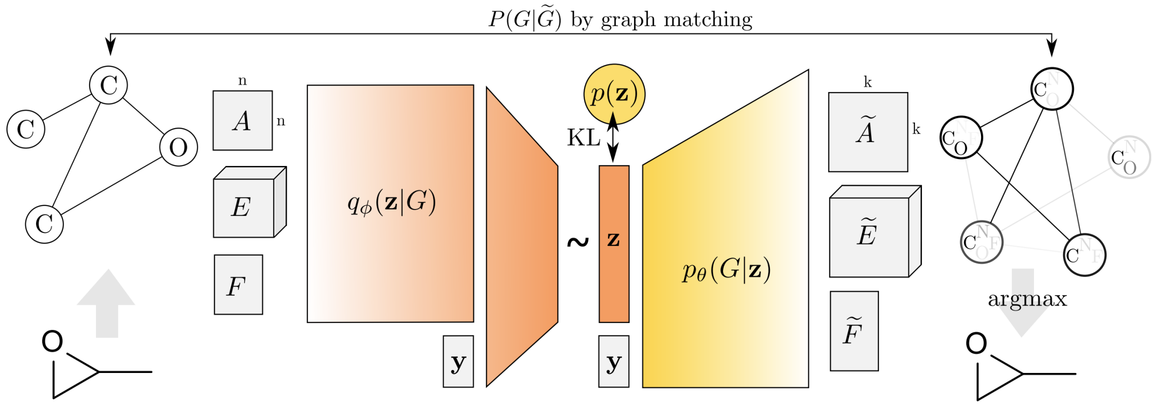


Figure 5.1: The overall process of GraphVAE. [3]

In figure 5.1, we show the general overall process of how GraphVAE works. The basic idea is by encoding the graph structure directly by its digital format using adjacency matrix $A \in [\mathbf{0}, \mathbf{1}]^{n\times n}$, edge attribute matrix E, and F being the node attribute matrix. Each metric can represent different aspect of the graph. For example, adjacency matrix indicates the connection between nodes within the graph, where $A_{i,j} = 1$ if node i and j is connected, and $A_{i,j} = 0$ if node i and j is disconnected. For edge attribute matrix E, in the context of drug discovery molecule generation, it can represent the bond information, e.g., whether the bond is a single-bond, double-bond or a σ-bond, etc. In node attribute matrix F, it can generally represent the atom attribute, for example, whether the atom is carbon or oxygen, etc., in the application of molecule generation.

As the input to encoder is $G = (A, E, F)$, its output pair is noted as $(\tilde{A}, \tilde{E}, \tilde{F})$. Interestingly, it is worth noting that the $\tilde{A} \in [\mathbf{0}, \mathbf{1}]^{k\times k}$ might have different shape as $A \in [\mathbf{0}, \mathbf{1}]^{n\times n}$. While in general, $n <= k$. However, the decoder can be super ineffective when k gets super large. So the pre-defined maximum number of graph nodes will be required for the Graph Variational AutoEncoder to function properly.

For loss function, Graph Variational AutoEncoder uses the negative log-likelihood $-\log(p_\theta(G))$:

$$\mathcal{L}(\phi, \varepsilon; G) = \mathbb{E}_{q_\phi(z|G)}[-\log p_\theta(G|z)] + KL[q_\theta(z|G)||p(z)], \tag{5.1}$$

where the first term in $\mathcal{L}$ is the reconstruction loss, minimizing the regression error. The second term is the KL divergence regularize the code space to allow the re-sample from re-parametrized distribution later.

We now look into details of the loss, while $G = (A, E, F)$, we can expand the $-\log p_\theta(G|z)$ by

$$-\log p_\theta(G|z) = -\lambda_A \log p(A'|z) - \lambda_E \log p(E|z) - \lambda_F \log p(F|z). \tag{5.2}$$

However, this method is limited by the pre-defined maximum number of nodes in the molecule graphs. And the maximum number should be small in general. This is due the growth of GPU memory and the number of the parameters. The space complexity is $\mathcal{O}(k^2)$ and the time complexity is $\mathcal{O}(k^4)$. This level of complexity is almost prohibitive in the large scale application. However, for many application, small graph generation will suffice so the method might still be worth studying.

We discussed yet another graph variational autoencoder method in this chapter named GraphVAE : Graph Variational AutoEncoder. This method takes in graph directly instead of a sequence of grammar or syntax rule embeddings. The method is also regularized by a pre-defined human prior on maximum graph node number. The effectiveness of this method has been observed particularly on smaller molecule graph generation.

CHAPTER 6

EXPERIMENTS AND RESULTS

In this section, we present experiments on various applications across multiple VAE methods. In section 6.1, we will start this chapter by presenting the experimental results on molecule reconstruction performance on different Variational AutoEncoders (VAEs). We will show the representation learning power gain from encoding the structure information comparing with pure text-based VAE in section 6.2. We will conclude the chapter by discussing pros and cons for different methods observed from experiments.

6.1 Molecule Reconstruction

In this section, the comparison among four Variational AutoEncoders (VAEs) [1, 2, 3, 33] discussed in previous chapters is presented with application in the molecule reconstruction. Basically, the initial valid molecule is fed into the encoder of the VAE and the output molecule from the decoder is fetched as result. The performance is measured by comparing the output molecule's the validity and consistency with the original input molecule.

Setup The result reported here is trained on ZINC drug-like datasets [34] and uses the train/test split in [1]. ZINC is a free database of commercially-available compounds for virtual screening. The drug-like dataset from ZINC contains 18,691,354 molecular SMILE representations.

Metrics For validation metrics, the ratio between valid generated molecules and total generated molecules is reported as **Validity**, i.e.,

$$Validity = \frac{\text{number of valid generated molecules}}{\text{total number of generated molecules}}. \tag{6.1}$$

For reconstruction metrics, the exact match accuracy (**EM Accuracy**) is reported. EM Accuracy [21] is essentially the ratio between exactly matched input/output molecule pairs and the total number of input molecules, i.e.,

$$EM\_Accuracy = \frac{\text{number of exactly matched molecules}}{\text{total number of tested molecules}}. \tag{6.2}$$

Due to the randomness in molecule generation, the result is reported as an accumulative value of 10 encode/decode runs.

Table 6.1: The comparison of Validity among different VAE methods.

GVAE [1]	SDVAE [2]	GraphVAE [33]	JTVAE [3]
7.2%	43.5%	13.5%	**100.0%**

Table 6.2: The comparison of EM accuracy among different VAE methods.

GVAE [1]	SDVAE [2]	JTVAE [3]
53.7%	76.2%	**76.7%**

Results and Analysis In table 6.1, the validity metric for each of the four variational auto-encoders (VAEs) is reported. It is clear that Junction True Variational AutoEncoder (JTVAE) outperforms all other methods. The Grammar Variational AutoEncoder (GVAE) only uses grammar syntax of SMILE sequence. This is assumed to be the least structure information usage among the four methods, which leads to the worse validity result here. Syntax-Directed Variational AutoEncoder (SDVAE) applies the additional constrain on molecule decoder over GVAE. This hence yields to a decent improvement over GVAE. GraphVAE is in a different method branch which only uses maximum graph vertex (or pre-defined max atom number in molecule) as structural information. So the validity performance is still worth than the SDVAE. Finally, the winner JTVAE here uses almost all of the information above and hence has the best performance.

In table 6.2, three out of the four VAE methods are compared for reconstruction [1]. One can easily observe that the Junction Tree Variational AutoEncoder outperforms all other methods in terms of reconstruction accuracy due to its exhaustive use of structural information. Comparing with Grammar Variational AutoEncoder (GVAE), its decode has the syntax prior to ensure the validity of the generated molecule and hence leads to higher chance for exactly matching in terms of reconstruction. Comparing with Syntax-Directed Variational AutoEncoder (SDVAE), its first step of using bird-view information of the molecule helps correct the overall architecture of generated molecule. This hence results in better reconstruction accuracy.

6.2 Ablation Study of Structural Information Fusion in Molecule Graph

Setup The experiment is conducted to ablatively study whether the structural/grammar information could help concentrate the information in the molecule latent space [1]. To serve this purpose, the result reported here is trained on ZINC drug-like datasets [34]. ZINC is a free database of commercially-available compounds for virtual screening. The drug-like dataset from ZINC contains 18,691,354 molecular SMILE representations. For the ease of visualization, the VAE model latent space dimension is set to 2.

[1]GraphVAE result is missing here due to implementation difficulty.

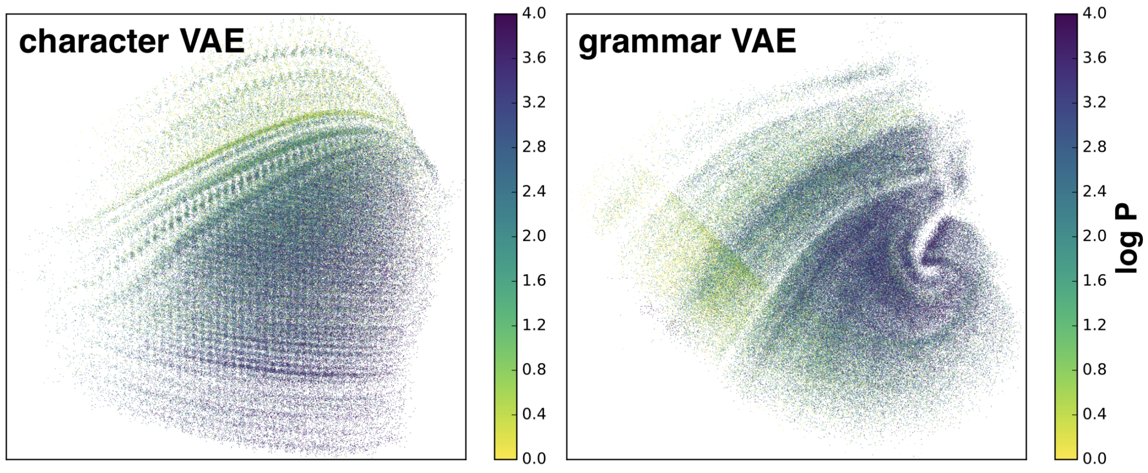


Figure 6.1: The CVAE [4] vs GVAE [1] latent space visualization. The color depth represents the LogP value of the corresponding molecule in the latent space.

Results In figure 6.1 (left), it is shown that, in Character Variational AutoEncoder (CVAE) which is a text-based encoding method for molecule graph, the lower right portion of the latent space tends to have higher LogP value. However, the molecule distribution is still sparse and seems to have multiple gap/holes in the latent space. In this case, if we sample molecule around a high LogP candidate using Monte Carlo method, we are likely getting into the holes and end up a much lower LogP candidate which is likely to be semantically invalid.

On the other hand, if we look at the 6.1 (right), the high LogP value area is more concentrated and tangled. If sampled from the high logP area, it is more likely to hit more valid high LogP candidates.

Analysis Overall, the Grammar Variational AutoEncoder (GVAE), when trained properly, can generate a more meaningful latent space in terms of chemical properties, like LogP value than Character Variational AutoEncoder (CVAE). The CVAE latent space is more sparse than GVAE one. This is likely due to the lack of grammar information of the molecule, which leads to higher probability in invalid molecule generation. These invalid molecules can form the holes and gaps in the latent space as shown in figure 6.1.

CHAPTER 7

CONCLUSIONS

In this paper, we survey modern deep learning on graphs techniques with application in drug candidate generation task. The drug candidate generation task is challenging due to the structural nature of the graph data input. How to incorporate the structural information into modeling is yet another open question.

Within a couple of available methods [35, 36], we select methods that fall in the range of variational autoencoder (VAE). We started our introduction gradually from a baseline VAE method by encoding only the grammar information of the graph into the modeling, namely GVAE. This method does not ensure the syntax correctness in the decoder. The gap was filled in another method named Syntax-Directed Variational AutoEncoder (SDVAE) by correcting syntax directly in the decoder. These methods have been improving in the direction of drug candidate valid ratio. Then, the method has been improving using various method by encoding different structural, human prior information. For example, GraphVAE encodes the maximum graph vertex number information into the graph generation. Junction Tree Variational AutoEncoder (JTVAE) takes a two-step approach. The first step is to generate a junction tree structured macro graph based on the sub-components, and then the detail of each sub-component is generated. This method ensures 100% valid generation of chemical compounds. We will then break down the conclusion further for each of the method mentioned above.

Grammar Variational AutoEncoder In this method [1], the authors proposed to represent the molecule graph as parse trees, which clearly outperforms text-based representation. It can be extended to multiple other areas like representation learning, optimization and also in a boarder range of applications where the atom data can be represented in a context-free grammar, e.g., programming language.

Syntax-Directed Variational AutoEncoder In this paper, the major novelty of SD-VAE [2] has been formed as a new method to improve the valid rate of generation by infusing both syntax and semantic constraints in generative model on graphs. Within the method, the authors introduce the *stochastic lazy attribute* to perform the offline syntax and semantic check when guiding online stochastic generation. Empirically, SDVAE present consistently improvement over the previous models in terms of validation accuracy, while the computational costs remain modestly same as before.

JTVAE: Junction Tree Variational AutoEncoder In JT-VAE, the authors propose a two-stage approach for drug candidate molecule graph generation. This method significantly improve the valid generation rate for drug candidates, outperforming most of the previous works. The authors would like to explore general low-tree-width graphs as a future direction.

GraphVAE: Towards Generation of Small Graphs Using Variational AutoEncoder In this paper, the authors proposed Graph Variational AutoEncoder and approach the decoder part to address the problem that generates the molecule graph from a continuous latent variable. The method itself encodes the human prior of maximum graph size into the modeling and evaluate the effectiveness of priors on two different sets. Experiments have shown reasonable quality on small molecule graphs.

So far, we have been discussing a couple of variational autoencoder based drug generative methods. However, in the deep learning on graphs research area, there are a few other models available for graphs that might be worth further studying, e.g., sequence-based generation methods [21, 7], etc. Future exploration can also goes to more diverse set of data modality.

REFERENCES

[1] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar variational autoencoder," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1945–1954.

[2] H. Dai, Y. Tian, B. Dai, S. Skiena, and L. Song, "Syntax-directed variational autoencoder for structured data," *arXiv preprint arXiv:1802.08786*, 2018.

[3] W. Jin, R. Barzilay, and T. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," *arXiv preprint arXiv:1802.04364*, 2018.

[4] R. Gómez-Bombarelli, D. Duvenaud, J. M. Hernández-Lobato, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *arXiv preprint arXiv:1610.02415*, 2016.

[5] D. Weininger, "Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules," in *Proc. Edinburgh Math. SOC*, vol. 17, 1970, pp. 1–14.

[6] G. Neglur, R. L. Grossman, and B. Liu, "Assigning unique keys to chemical compounds for data integration: Some interesting counter examples," in *International Workshop on Data Integration in the Life Sciences*. Springer, 2005, pp. 145–157.

[7] X. Zhang, S. Wang, F. Zhu, Z. Xu, Y. Wang, and J. Huang, "Seq3seq fingerprint: Towards end-to-end semi-supervised deep drug discovery," in *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. ACM, 2018, pp. 404–413.

[8] Y. Hu, E. Lounkine, and J. Bajorath, "Improving the search performance of extended connectivity fingerprints through activity-oriented feature filtering and application of a bit-density-dependent similarity function," *ChemMedChem*, vol. 4, no. 4, pp. 540–548, 2009.

[9] R. C. Glen, A. Bender, C. H. Arnby, L. Carlsson, S. Boyer, and J. Smith, "Circular fingerprints: flexible molecular descriptors with applications from physical chemistry to adme," *IDrugs*, vol. 9, no. 3, p. 199, 2006.

[10] H. Morgan, "The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service," *J. Chemical Documentation*, vol. 5, pp. 107–113, 1965.

[11] D. Rogers and M. Hahn, "Extended-connectivity fingerprints," *Journal of chemical information and modeling*, vol. 50, no. 5, pp. 742–754, 2010.

[12] N. M. O'Boyle, C. M. Campbell, and G. R. Hutchison, "Computational design and selection of optimal organic photovoltaic materials," *The Journal of Physical Chemistry C*, vol. 115, no. 32, pp. 16 200–16 210, 2011.

[13] C. Rupakheti, A. Virshup, W. Yang, and D. N. Beratan, "Strategy to discover diverse optimal molecules in the small molecule universe," *Journal of chemical information and modeling*, vol. 55, no. 3, pp. 529–537, 2015.

[14] G. Subramanian, B. Ramsundar, V. Pande, and R. A. Denny, "Computational modeling of β-secretase 1 (bace-1) inhibitors using ligand based approaches," *Journal of Chemical Information and Modeling*, vol. 56, no. 10, pp. 1936–1949, 2016.

[15] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande, "Moleculenet: a benchmark for molecular machine learning," *Chemical Science*, vol. 9, no. 2, pp. 513–530, 2018.

[16] J. Gomes, B. Ramsundar, E. N. Feinberg, and V. S. Pande, "Atomic convolutional networks for predicting protein-ligand binding affinity," *arXiv preprint arXiv:1703.10603*, 2017.

[17] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," *arXiv preprint arXiv:1801.03226*, 2018.

[18] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*, 2016, pp. 2014–2023.

[19] R. Li and J. Huang, "Learning graph while training: An evolving graph convolutional neural network," *arXiv preprint arXiv:1708.04675*, 2017.

[20] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in neural information processing systems*, 2015, pp. 2224–2232.

[21] Z. Xu, S. Wang, F. Zhu, and J. Huang, "Seq2seq fingerprint: An unsupervised deep molecular embedding for drug discovery," in *BCB*, 2017.

[22] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *European conference on computational learning theory.* Springer, 1995, pp. 23–37.

[23] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[24] T. K. Ho, "Random decision forests," in *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, vol. 1. IEEE, 1995, pp. 278–282.

[25] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[26] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[27] S. Wang, J. Yao, Z. Xu, and J. Huang, "Subtype cell detection with an accelerated deep convolution neural network," in *International Conference on Medical Image Computing and Computer-Assisted Intervention.* Springer, 2016, pp. 640–648.

[28] Z. Xu and J. Huang, "Detecting 10,000 cells in one second," in *International Conference on Medical Image Computing and Computer-Assisted Intervention.* Springer, 2016, pp. 676–684.

[29] J. K. Baker, "Trainable grammars for speech recognition," *The Journal of the Acoustical Society of America*, vol. 65, no. S1, pp. S132–S132, 1979.

[30] T. L. Booth and R. A. Thompson, "Applying probability measures to abstract languages," *IEEE transactions on Computers*, vol. 100, no. 5, pp. 442–450, 1973.

[31] M. Johnson, T. L. Griffiths, and S. Goldwater, "Adaptor grammars: A framework for specifying compositional nonparametric bayesian models," in *Advances in neural information processing systems*, 2007, pp. 641–648.

[32] M. Rarey and J. S. Dixon, "Feature trees: a new molecular similarity measure based on tree matching," *Journal of computer-aided molecular design*, vol. 12, no. 5, pp. 471–490, 1998.

[33] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," in *International Conference on Artificial Neural Networks.* Springer, 2018, pp. 412–422.

[34] J. J. Irwin, T. Sterling, M. M. Mysinger, E. S. Bolstad, and R. G. Coleman, "Zinc: a free tool to discover chemistry for biology," *Journal of chemical information and modeling*, vol. 52, no. 7, pp. 1757–1768, 2012.

[35] Z. Xu and J. Huang, “A general efficient hyperparameter-free algorithm for convolutional sparse learning.” in *AAAI*, 2017, pp. 2803–2809.

[36] S. Wang, Z. Xu, C. Yan, and J. Huang, “Graph convolutional nets for tool presence detection in surgical videos,” in *International Conference on Information Processing in Medical Imaging.* Springer, 2019, pp. 467–478.

BIOGRAPHICAL STATEMENT

Siyu Xiu received her M.S. in Computer Science from the University of Texas at Arlington at 2019. Prior to beginning the M.S. program, Siyu obtained her B.S. degree from Northeast Forestry University, China in 2015 in Geography Information System.