Study of Deep Generative Adversarial Network Generation of Molecular Graphs

Anthony Vasquez
Johns Hopkins University
Whiting School of Engineering
Baltimor, MD
avasque1@jh.edu

Abstract—

Deep Generative Adversarial Networks (DGANs) have demonstrated remarkable effectiveness across various fields, producing high-quality samples that can often deceive both humans and neural networks. For instance, DGANs have been employed in anomaly detection within banking, creating synthetic driving environments for autonomous vehicles, generating synthetic audio and facilitating audio style-transfer, restoring images, and many other applications. One topic of interest, is in the field of chemistry where the challenging problem of generating molecular graphs exists. Many solutions have been tried with varying degrees of success, such as using a variational auto-encoder (Bidisha, 2019). The work presented here compares synthetic GAN generations using recurrent and multi-head attention, named Synthetic Molecule GAN (SMGAN), to data synthesis of a MolGAN (Nicola De Cao, 2018). by training the networks to generate realistic molecular graphs similar to Simplified Molecular Input Line Entry System (SMILES), a database that generates string representations of molecules and reactions (National Library of Medicine, 2024).

Keywords - DGANS, SMILES, Medicine, Chemistry

Introduction

The generation of new molecules is of interest in medicine, where designing new drugs is long, costly, laborious, and yields few new molecular discoveries (Bidisha, 2019). Some estimates suggest that it takes an average of 10-15 years to develop a single new medicine and comes with a $2.6 dollar price tag (Phrma, 2024). There is much interest in deriving or automating a way to generate physically viable molecules which could potentially speed up the discovery-to-shelf pipeline of pharmaceuticals at a fraction of the price.

Among the many difficulties with working with molecular data, is that it is not readily consumable by machine learning algorithms. This is the case with most data, for example Natural Language Processing (NLP) and the use of term frequency-inverse document frequency (TF-IDF), which is a well-known encoding algorithm (Das, 2018). However, there are few vectorizations transforms for molecular data. One molecular vectorizer is SMILES (Weininger, 1987), a flexible fragment based molecular representation framework. SMILES uses various characters in a character sequence which conveys relationships among the different atoms in molecules. Another much newer approach designed to extend complexity traditional SMILES strings is that of t-SMILES. This string representation contains additional information and by increasing the number of characters used to convey more relational information (Jaun-Ni Wu, 2024). For the sake of simplicity, and as a way to make this work more relevant to previous work on generative molecule synthesis using GANs, the SMILES representation was used. In addition, many tools have been created and extensively tested on SMILES strings over many years so is expected to be more stable. The SMILES representation captures the molecular graph which is used to create a representative vector representation as seen in NLP. In this work, a basic GAN is used to generate molecules that are in some cases, valid molecules, novel, unique, soluble, druglike, and synthesizable.

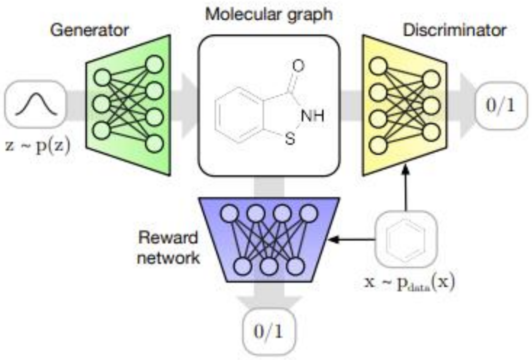
Related work

Various methods have been proposed, but two generative methods stand out that explore similar strategies. One method is the VAE, from which many models have been implemented, such as, NeVAE, GraphVAE (Simonosvsky, 2018), GrammarVAE (Kusner, 2017), CVAE (Pagnomi, 2018), SDVAE (Hankun Dai, 2018), JTVAE (Jin, 2019), and CGVAE (Rigoni, 2020), to name a few. This is by no means an exhaustive list, and it can be safely assumed that newer models have been explored. For the sake of brevity, only NeVAE will be discussed, as it is well documented.

The authors of NeVAE claim, "[their] model guarantee[s] a set of valid generated molecules", and their model can "discover plausible, diverse and novel molecules more effectively than several state-of-the-art methods" (Bidisha, 2019). An innovation used in their experimentations was to use a VAE conjunction with a data representation that captures structural properties of a molecule. Their assumption is that the vector representation embeds N molecular graphs, composed of various sets of atoms (nodes) and bonds (edges), where each may contain various lengths; $\{G_i = (V_i \epsilon_i)\}_{i \epsilon [N]}$, where V_i and ε_i are nodes and edges respectively, there exists a set of features $F = \{f_u\}_{u \epsilon V}$ and edge weighs, $Y = \{y_{uv}\}_{(u,v) \epsilon \varepsilon}$, that can be learned (Bidisha, 2019). Their technique creates a continuous latent space, enabling users to extract useful and novel chemical compounds (Bidisha, 2019). Their model is evaluated by decoder generation quality metrics, namely, Novelty, Uniqueness, and Validity. These metrics will be discussed in more depth in Evaluation Metrics.

The other method uses a different deep generative paradigm is that of MolGAN, a generative adversarial approach. MolGAN combines reinforcement learning (RL) by rewarding an agent with high objective scores based on several metrics (Nicola De Cao, 2018). Among the objective scores of this method are adversarial loss, to measure the quality of the generated images from the real ones, a loss for molecule solubility, and drug-likeness (Nicola De Cao, 2018). Additionally, an RL component is trained to learn the quality of generated molecules based on "desired chemical properties" (Nicola De Cao, 2018). An objective loss is also calculated for the RL agent during training. The GAN approach to novel molecule generation is the approach used in this work, but there are some key differences that will be expanded upon in the network section of this paper.

Figure 1. The MolGAN Network has three main components, a generator, discriminator, and an RL component.



Data set:

SMILES is a methodology and database, where molecular graph theory is combined with natural language processing to produce unique character strings, which explain the underlying structure of molecules, such as, atoms, bonds, and connectivity (Weininger, 1987). In this work, a small sample of the SMILES dataset was used. Of the 249,456 molecular strings, only about or 25 percent of the dataset was used to train. It is unclear of whether adding more data would significantly improve model convergence, but this amount of data is mostly arbitrary and influenced by time-to-train constraints. These samples were randomly sampled from the entire dataset using scikit-learn train-test-split method (Pedregosa, 2011).

Preprocessing

The SMILES strings can be converted to bit-map representations for numeric modeling and nearest neighbor modeling of other molecules. However, because bit-map conversion is not lossless, it is difficult to cross reference between other SMILES molecules. A bitmap cannot be directly converted back to a SMILES string for example, which means that indirect methods need to be used for molecule comparisons, such as similarity metrics. For that reason, the SMILES strings were not converted to bitmaps, but were instead tokenized.

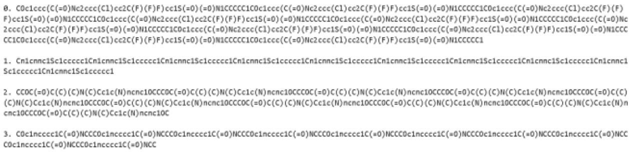
Each of the train set molecule strings were tokenized by using regular expressions to split the string into segments by using special characters. For example, “-” is a single bond, “=” is a double bond, and “#” is a triple bond. More symbols can be found in Table 1.

Table 1. This table shows common SMILES symbols (Kim, 2017).

Single Bond	-
Double Bond	=
Triple Bond	#
Arom. Bond	:
Pos. Charge	[C+]
Neg. Charge	[C-]
Arom. Carbon	c (lowercase)

The original strings are characterized by each sample no spaces, letters representing atoms, and special characters that relate the relationship of an atom to other atoms in their respective molecules. An example of this notation is found in the below Table 2, which shows four examples of SMILES molecules.

Table 2. Example of SMILES Strings



```
0. COc1ccc(C(=O)Nc2ccc(Cl)cc2C(F)(F)F)cc1S(=O)(=O)N1CCCCC1COc1ccc(C(=O)Nc2ccc(Cl)cc2C(F)(F)F)cc1S(=O)(=O)N1CCCCC1COc1ccc(C(=O)Nc2ccc(Cl)cc2C(F)(F)
F)cc1S(=O)(=O)N1CCCCC1COc1ccc(C(=O)Nc2ccc(Cl)cc2C(F)(F)F)cc1S(=O)(=O)N1CCCCC1COc1ccc(C(=O)Nc2ccc(Cl)cc2C(F)(F)F)cc1S(=O)(=O)N1CCCCC1COc1ccc(C(=O)Nc
2ccc(Cl)cc2C(F)(F)F)cc1S(=O)(=O)N1CCCCC1COc1ccc(C(=O)Nc2ccc(Cl)cc2C(F)(F)F)cc1S(=O)(=O)N1CCCCC1COc1ccc(C(=O)Nc2ccc(Cl)cc2C(F)(F)F)cc1S(=O)(=O)N1CCC
CC1COc1ccc(C(=O)Nc2ccc(Cl)cc2C(F)(F)F)cc1S(=O)(=O)N1CCCCC1COc1ccc(C(=O)Nc2ccc(Cl)cc2C(F)(F)F)cc1S(=O)(=O)N1CCCCC1

1. Cn1cnnc1Sc1ccccc1Cn1cnnc1Sc1ccccc1Cn1cnnc1Sc1ccccc1Cn1cnnc1Sc1ccccc1Cn1cnnc1Sc1ccccc1Cn1cnnc1Sc1ccccc1Cn1cnnc1Sc1ccccc1Cn1cnnc1Sc1ccccc1Cn1cnnc1
Sc1ccccc1Cn1cnnc1Sc1ccccc1

2. CCOC(=O)C(C)(C)N(C)Cc1c(N)ncnc1OCCCOC(=O)C(C)(C)N(C)Cc1c(N)ncnc1OCCCOC(=O)C(C)(C)N(C)Cc1c(N)ncnc1OCCCOC(=O)C(C)(C)N(C)Cc1c(N)ncnc1OCCCOC(=O)C(C)
(C)N(C)Cc1c(N)ncnc1OCCCOC(=O)C(C)(C)N(C)Cc1c(N)ncnc1OCCCOC(=O)C(C)(C)N(C)Cc1c(N)ncnc1OCCCOC(=O)C(C)(C)N(C)Cc1c(N)ncnc1OCCCOC(=O)C(C)(C)N(C)Cc1c(N)n
cnc1OCCCOC(=O)C(C)(C)N(C)Cc1c(N)ncnc1OC

3. COc1ncccc1C(=O)NCCCOc1ncccc1C(=O)NCCCOc1ncccc1C(=O)NCCCOc1ncccc1C(=O)NCCCOc1ncccc1C(=O)NCCCOc1ncccc1C(=O)NCCCOc1ncccc1C(=O)NCCCOc1ncccc1C(=O)NCC
COc1ncccc1C(=O)NCCCOc1ncccc1C(=O)NCC
```

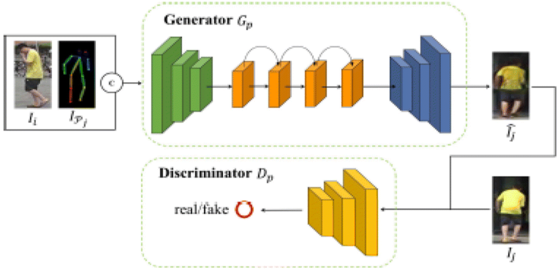
The SMILES strings were tokenized by using regular expressions to split the strings by symbol, for example, $pattern = r"(\backslash[[\text{^}\backslash[\backslash]] *\backslash])"$ for each SMILES string. This encoding allows an embedding layer to be used when modeling.

Architecture

The architecture used here consists of a generative model with adversarial discriminator and generator networks. The GAN architecture uses embedding, recurrence, multi-head-attention, and fully connected layers. Each of these sub-architectures play an important role in generating novel molecules, and each relies on various hyperparameters further described in Hyperparameter Tuning. For now, a brief introduction is given to each of these layers.

The GAN architecture was discovered by Ian Goodfellow in 2016, when he described a model consisting of a sample generator G_θ and discriminator D_φ networks that trained according to an adversarial policy (Goodfellow, 2014). G_θ attempts to generate realistic looking samples and D_θ attempts to classify if the image came from the distribution of real data or not. At each iteration, the network that loses updates the weights based on the real data distribution. The GAN’s network optimizes according to its respective loss function, and the model is said to converge when G_θ can no longer fool D_φ.

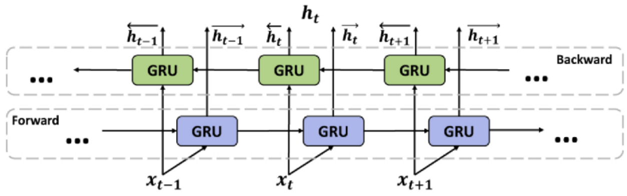
Figure 2. Basic GAN Architecture (Alqahtani, 2019)



Though GANs were innovative and fast adopted, there was one big problem – mode collapse (Mangalam, 2021). Mode collapse is when the generator fails to produce diverse samples. To combat this, a major improvement to the GAN was proposed with the contributions of WGAN (Arjovsky, 2017). In this paper stability issues were improved on by updating the loss function to the Wasserstein loss, weight clipping, and gradient regularization (Arjovsky, 2017).
For this reason, the WGAN paradigm was used in this work to address potential stability.

The Gated Recurrent Unit (GRU) is a simplified version of a Long-Short-Term-Memory network (LSTM) with less gating and better time complexity. For example, the GRU only has two gates, a reset gate that allows the model to determine how much of a previous hidden state to use, and an update gate that determines how much of a previous hidden state to pass to the next timestep. GRUs have a bidirectional mechanism, that allows it to learn sequences from start to end, and end to start in parallel. We will call this version of GRU the BiGRU, which is how it will be referred to moving forward.

Figure 3. The BiGRU is able to learn sequential data by maintaining memory. Unlike traditional RNN, the BiGRU attempts to use data from both the past and future directions. Put another way, it attempts to learn the sequence from beginning to end and end to beginning simultaneously (Liu, 2021).


h_t
$\overleftarrow{h}_{t-1}$
$\overrightarrow{h}_{t-1}$
$\overleftarrow{h}_t$
$\overrightarrow{h}_t$
$\overleftarrow{h}_{t+1}$
$\overrightarrow{h}_{t+1}$
Backward
Forward
GRU
x_{t-1}
x_t
x_{t+1}

Multi-head-attention is an extension of self-attention that allows the network to focus its attention on segments of the input data. For example, a multi-head attention network with four-heads means that the data is attended by four attention mechanisms, to take advantage of both syntactic and semantic features learned from the linear projections of each segment of a sequence (Cordeonnier, 2021). For this work, it is currently unknown of how much this step contributes to the generation of novel molecules, but given the success of self-attention and multi-head attention, it is assumed that it has a positive impact on both training and generation modes of the network.

Hardware/Software

The operating system was Linux, using CUDA 12.2 (Nickolls, 2008), 2 NVIDIA RTX A4000s, with 24 logical processors. The logical processors proved to very helpful with parallelizing metric calculations, which are discussed in more detail in Evaluation Metrics. Two GPUs were helpful with training larger batch-sizes, but they did not contribute much in speeding up training, since using a GRU is a sequential task which clock speeds are more important.

Initial Training

Binary cross entropy loss (BCEL) was used for training, along with the Adam optimizer (Pytorch, 2024). A small learning rate of 0.00025 for both generator and discriminator networks were used at the start of training. A basic learning rate scheduler was used to update the learning rate with $\gamma = 0.1$ (update coefficient) incremented at every five epochs. The momentum moving average was set to 0.5 to 0.99 with L2 regularization of 1e-5.

All layers contained dropout with a probability of 0.25 with the exception of the fully connected layers which implemented zero dropout probability none. The embedding dimension was set to the length of the vocabulary by an embedding dimension of 32, an arbitrarily picked value to be tuned later. The hidden dimension was set to 64, which was used in the GRU, attention and fully connected layers. However, the BiGRU required a doubling of hidden dimension size, and therefore an increase in time complexity. As mentioned previously, WGAN methodology was used during training and the weights were clipped to ± 0.01 at each batch iteration. Furthermore, the discriminator was allowed to optimize three times for every one generator update.

The network was trained for 50 epochs and both generator and discriminator losses were monitored every two epochs for signs of mode collapse. For example, if either the generator or discriminator displayed signs of exploding or diminishing gradients, the experiment was stopped. However, training never stopped while using the WGAN paradigm.

Hyperparameter Tuning

Tune is a scalable and flexible hyperparameter tuning API (Ray, 2024). It was used for hyperparameter tuning for its robust logging functionality and ease of use. Using Ray Tune, the Asynchronous Successive Halving Algorithm (ASHA) (Ray, 2024) was used to search a hyperparameter grid with predetermined boundaries. The ASHA tuning schedule was shown to be one of the most effective tuning schedulers, as effectively implements successive halving and asynchronous execution using a Rung method (Li, 2018). An added benefit of using Ray Tune is that it schedules the experiments, automatically juggling hardware resource. For example, the resources per experiment parameter was set to 2 CPUs and 1 GPU per run, with a maximum concurrent trial of four experiments at any one time.

The hyperparameter search space initially started with the Adam, NAdam, and RMSprop optimizers, but was reduced to just RMSprop because it tended to converge to a lower generator loss faster than the others. Other searched hyperparameters include, generator learning rate, discriminator learning rate, batch size, hidden layer dimension, embedding dimension, number of GRU layers, dropout probability, number of attention heads, bidirectional, number of extra discriminator updates, weights clip value, and max number of epochs: 25. The search was conducted with a max number of epochs of 25 with only ten percent of the data, and a total of 300 epochs.

Table 3. Hyperparameter Search Space

Hyperparameter	Search Space	Selection
Optimizer	Adam, Nadam, RMSProp	Choice
Learning Rate	0.00002 - 0.002	Loguniform
Batch Size	32, 64, 128, 256, 384	Choice
Hidden Layer Dim	16, 32, 64, 128	Choice
Embedding Dim	16, 32, 64	Choice
Num GRU Layers	2, 3	Choice
Dropout Prob	0.25, 0.5	Choice
Num Attn. Heads	2, 4, 8	Choice
Bidirectional	True, False	Choice
Num Discr. Updates	1, 2, 3, 4, 5	Choice
Weight Clip Value	0.01 - 0.001	Loguniform

As previously mentioned, an ASHA scheduler was also used for efficient hyperparameter tuning. The parameters used included loss metric: generator loss, mode: minimum, maximum time to train of 2500 seconds, an early stopping grace period of 5, and a reduction factor of 2. The reduction factor determines which hyperparameters make it to the next round using a rung policy.

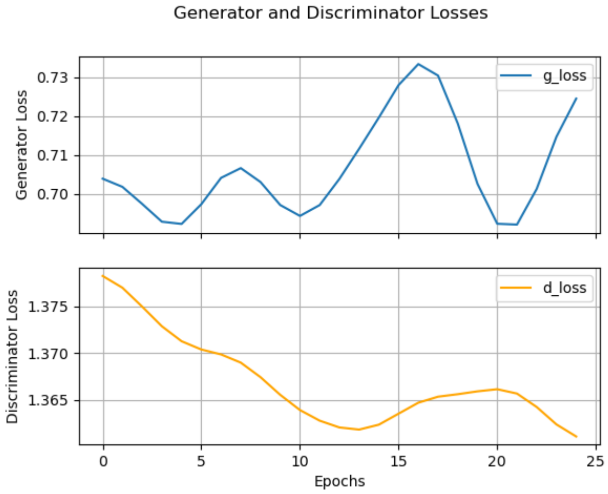
Table 4. Tuned Hyperparameters

Hyperparameter	Value
Optimizer	RMSProp
Gen. LR	0.00408680002539221
Disc. LR	0.0134480709026601
Batch Size	128
Hidden Layer Dim	128
Embedding Dim	32
Num GRU Layers	2
Dropout Prob	0.5
Num Attn. Heads	4
Bidirectional	TRUE
Num Discr. Updates	3
Weight Clip Value	0.00810198750825

Final Training

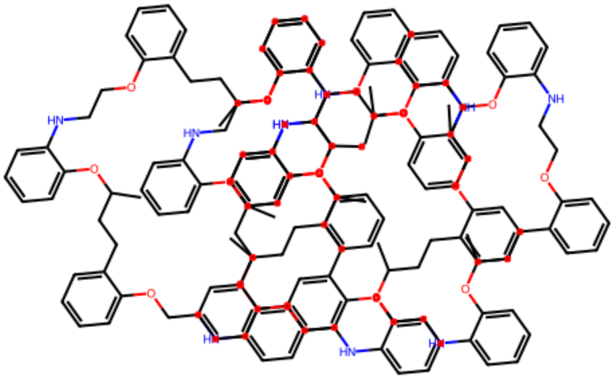
The model configured with the optimal hyperparameters was trained on 25 percent of the full dataset (62,500 samples). Both generator and discriminator losses were monitored for signs of instability or mode collapse. The discriminator loss appeared to be more stable than the generator loss, but both stayed within a small range, implying that the networks match each other's performance. Furthermore, there was a lack of evidence of mode collapse. to better understand how well the networks generalized, more training should be done with this same configuration. See Figure 4

Figure 4. Shown here are generator and discriminator training losses.



Using the trained network and a maximum generator output length of 10, the network can generate valid molecules as seen in Figure 5.

Figure 5. Valid and unique SMGAN generated image.



Inference Tuning

The best trained model was used to run a simple loop over various values of maximum length of generator output. Though the model was optimized for a generator max length output of 10, that does not rule out the possibility that lengths smaller or larger than this number could produce more viable molecules. Given hardware restraints, the values picked were the counting numbers in the interval [3, 14].

Evaluation Metrics

A molecule's viability was judged by using several performance metrics also used in MolGAN (Nicola De Cao, 2018). These metrics include generator BCEL, generated molecule validity, uniqueness, novelty, solubility druglikeness, and synthesizability. In addition, the valid SMILES strings were inspected and subsequently plotted. All of the molecular descriptors for calculating solubility, druglikeness, and synthesizability were obtained from RDKit.

Validity: is the ratio of the number of valid molecules to the number of all generated molecules. A molecule is said to be valid if it is recognized as valid by an open-source cheminformatics software (RDKit, 2024).

Uniqueness: is the percentage of unique generations to all valid generations. This metric attempts to quantify the diversity of molecule generations (Nicola De Cao, 2018).

Novelty: is the ability of the network to generate non-existent samples, that are never seen by the model. The generated SMILES string is canonicalized, then compared to other known molecules (RDKit, 2023). If the molecule exists but is not in-sample, then it is said to be novel.

Solubility: is an approximation of the degree of molecule hydrophilicity, such as high reactivity to water or dissolvability (Comer, 2001).

Druglikeness: Is calculated using the Quantitative Estimate of Druglikeness (QED) (Bickerton, 2012). A high druglikeness score is $QED \geq 0.7$, whereas a value of 0.4 to 0.69 is considered moderate, and anything less is considered low (Bickerton, 2012).

Synthesizability: is a probabilistic estimation of the ease of synthesizing a molecule (Ertl, 2009).

Results

Molecules were generated using the optimal inference hyperparameters with the optimal generator max length of 13, followed by 10 and 14. This result shows evidence that better performance is possible with larger max generator output lengths, and should be further investigated in future work. For the purpose of this study, the best values were picked from any of the max length generations. Solubility, druglikeness, and synthesizability were all normalized between [0, 1].

Table 5. MolGAN and SMGAN Best Scoring Results

Metric	SMGAN	MolGAN
Valid (%)	17.50	100.00
Unique (%)	2.00	3.20
Novel (%)	100.00	100.00
Solubility	0.28	0.89
Druglikeness	0.04	0.62
Synthesizability	0.25	0.95

Though MolGAN uses a different dataset, both networks are able to generate valid, unique, novel, soluble, druglike, and synthesizable molecules. This is not a direct comparison, but rather a proof of concept of SMGAN's ability to create both novel and feasible molecules.

Figure 6. Valid SMGAN Generated Molecule w/ Max Length = 14

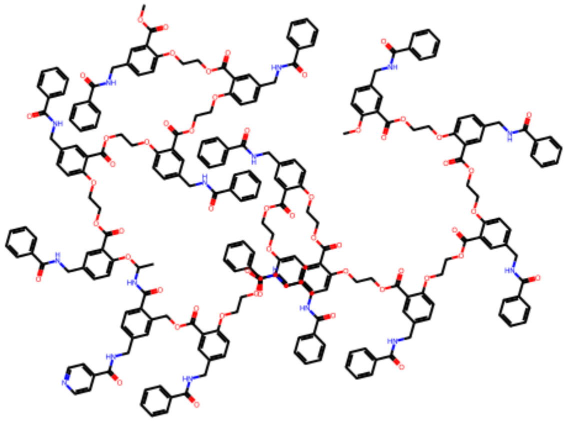
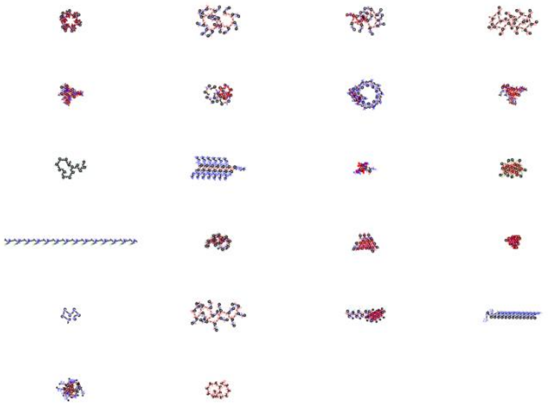


Figure 7. Several Valid and Unique SMGAN Generated Molecules with Max Length = 14.



Future Work

Besides time, another constraint is the hardware availability. More GPU memory could allow for optimization on longer string generations. As was shown in the section on inference tuning, the string length of the generator is an important hyperparameter, and though the network was optimized on a string length of 10, during inference, a length of 14 performed better. Unfortunately, 14 is the maximum string currently available given the hardware setup of this experiment.

Furthermore, training for more epochs should be investigated. SMGAN was trained for 25 epochs, whereas MolGAN trained for 300 epochs (Nicola De Cao, 2018). MolGAN also used 5k carefully picked molecules, rather than 62K randomly sampled molecules. Perhaps focusing in on certain molecules with a maximum or minimum size could help convergence. Other discriminators can be used such as mol weight, charge, bond count, and many others.

Another improvement is to add a loss function that considers one, some, or all of the viability metrics. Along the same grain, perhaps splitting the data into a training set that score high on most of, if not all of the viability metrics. Theoretically, that would result in positive performance gains to solubility, synthesizability, and druglikeness. Of course adding both of these viability methods would be ideal.

Conclusion

In this work, it was shown that valid SMILES strings could be generated synthetically through the use of SMGAN. With the best performing SMGAN model, generations that are non-valid are > five time more likely, however, some generations are valid, and score reasonably well according to the feasibility metrics defined in Evaluation Metrics.

References

Alqahtani, H. E. (2019, Dec 19). Applications of Generative Adversarial Networks (GANS): An Updated Review.

Arjovsky, M. E. (2017, Jan 26). Wasserstein GAN.

Bickerton, G. R. (2012). Quantifying the chemical beauty of drugs.

Bidisha, S. (2019). NeVAE: A Deep Generative Model for Molecular Graphs.

Comer, J. a. (2001). Lipophiicity profiles: theory and measurement. Zurick, Switzerland.

Cordeonnier, J.-B. E.-A. (2021, May 20). Multi-Head Attention: Collaborate Instead of Concatenate.

Das, B. (2018). An Improved Text Sentiment Classification Model Using TF-IDF and Next Word Negation.

Ertl, P. a. (2009). Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of Cheminformatics*, 1-8.

Goodfellow, E. A. (2014, June 10). Generative Adversarial Networks.

Hankun Dai, E. A. (2018, Febuary 24). Syntax-Directed Variational Autoencoder for Structured Data.

Jaun-Ni Wu, E. A. (2024). t-SMILES: A Fragment-based Molecular Representation Framework for De Novo Ligand Design. Hunan, China.

Jin, W. E. (2019, March 29). Junction Tree Variational Autoencoder for Molecular Graph Generation.

Kim, S. (2017). *Line Notation (SNUKES and InCHI)*. Retrieved from LibreTexts Chemistry.

Kusner, M. E. (2017, March 6). Grammer Variational Autoencoder.

Li, L. E. (2018, March 16). A System for Massively Parallel Hyperparameter Tuning.

Liu, X. E. (2021). Bi-directional gated recurrent unit neural network based nonlinear equalizer for coherent optical communication systems.

Mangalam, K. G. (2021, Dec 29). Overcoming Mode Collapse with Adaptive Multi Adversarial Training.

National Library of Medicine. (2024, July 22). *National Center for Biotechnology Information*. Retrieved from PubChem: https://pubchem.ncbi.nlm.nih.gov/

Nickolls, J. E. (2008, March 2). Scalable Parallel Programming with CUDA.

Nicola De Cao, T. K. (2018). MolGAN: An Implicit Generative Model for Small Molecular Graphs. Stochholm, Sweden.

Pagnomi, A. E. (2018, December 11). Conditional Variational Autoencoder for Neural Machine Translation.

Phrma. (2024, July 22). *Research & Development Policy Framework*. Retrieved from phrma.org: https://phrma.org/policy-issues/Research-and-Development-Policy-Framework

Pytorch. (2024, August 20). *Adam*. Retrieved from Pytorch: https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

Ray. (2024, 8 20). *Tune*. Retrieved from Ray: https://docs.ray.io/en/latest/tune/index.html

Ray. (2024, 8 20). *Tune Trial Schedulers*. Retrieved from Ray: https://docs.ray.io/en/latest/tune/api/schedulers.html

RDKit. (2023). *rdkit.Chem.rdMolTransforms module*. Retrieved from rdkit.org: https://www.rdkit.org/docs/source/rdkit.Chem.rdMolTransforms.html

RDKit. (2024, August 20). *RDKit: Open-Source Cheminformatics Software*. Retrieved from rdkit.org: 2024

Rigoni, D. E. (2020, Sep 1). Conditional Constrained Graph Variational Autoencoders for Molecule Design.

Simonosvsky, M. K. (2018, Feb 9). GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders.

Vaswani, A. E. (2017, June 12). Attention is All You Need.

Weininger, D. (1987, June 1987). SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules. Claremont, California, United States.