Generative Models for Structured Discrete Data with Application to Drug Discovery

Chenghui Zhou

CMU-ML-24-109
August, 2024

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Barnabas Poczos, **Chair**
Andrej Risteski
Tom Mitchell
Siamak Ravanbakhsh

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by Air Force Research Laboratory awards FA87501620042 and FA87501720130 and National Science Foundation award IIS1637927.

Keywords: Generative Models, Discrete Data, Drug Discovery, Variational Autoencoder, Diffusion Models, Machine Learning

For my father, who was my greatest supporter

Abstract

My thesis focuses on generative models and their applications to discrete data. We propose novel algorithms that integrate insights from state-of-the-art generative models and domain-specific knowledge of discrete data types. These algorithms aim to enhance property similarity to training data, improve data validity, and elevate the overall quality of generated outputs. The first part of my thesis investigates converting geometric images into a discrete representation using context-free grammar. We discuss effective and scalable techniques to identify suitable representations in a large search space. The second part of my thesis examines the behavior of Variational Autoencoders (VAEs) in recovering high-dimensional data embedded in lower-dimensional manifolds, assessing their ability to recover the manifold and the data density over it. Extending our exploration of VAEs into discrete data domains, particularly in molecular data generation, we found that a method enhancing VAEs' manifold recovery for continuous data also significantly improves discrete data generation. We study its benefits and limitations using the ChEMBL dataset and two smaller datasets of active molecules for protein targets. Lastly, addressing the challenge of generating stable 3D molecules, the thesis incorporates a non-differentiable chemistry oracle, GFN2-xTB, into the denoising process to improve geometry and stability. This approach is validated on datasets like QM9 and GEOM, demonstrating higher stability rates among generated molecules.

Acknowledgments

I am grateful to my advisor, Barnabas Poczos, for his invaluable guidance and support throughout the course of my research. His expertise, insightful feedback, and unwavering encouragement have been instrumental in shaping this thesis. He provided not only academic mentorship but also inspiration and motivation, helping me to navigate challenges and stay focused on my goals. His commitment to excellence and dedication to my growth as a researcher has profoundly impacted my academic journey.

I would like to extend my heartfelt thanks to Professor Andrej Risteski for his invaluable collaboration and support throughout this research. Working with him has been a truly enriching experience, and his expertise and insightful contributions have enhanced the quality of this thesis. In addition, I would also like to express my sincere appreciation to the other members of my thesis committee, Professor Tom Mitchell and Professor Siamak Ravanbakhsh. Their constructive feedback and thoughtful insights have been invaluable throughout the development of this thesis. Their dedication, and willingness to review my work have contributed to the refinement and depth of this research. Each of you has offered unique perspectives that have greatly enriched the overall quality of this thesis.

In addition to my thesis committee members, I have learned a great deal from my collaborators. Chun-Liang Li, with whom I collaborated on the program synthesis project in Chapter 2, taught me the importance of optimism and perseverance in advancing research. Frederic Koehler and Andrej deepened my understanding of fundamental questions in machine learning, significantly enhancing my literacy in theoretical papers. Viraj Mehta introduced me to JAX and helped me get familiar with it. When I started to explore the field of drug discovery, Professor Olexandr Isayev provided a critical dataset of active molecules for Chapter 4, and generously shared his expertise in the field. Professor Newell Washburn and his group expanded my knowledge of chemistry, with Sijie Fu offering key insights into quantum chemistry that aided the project in Chapter 5. Engaging discussions on the intersection of chemistry and machine learning with Amira Alakhdar, Arav Agarwal, and Euxhen Hasanaj were invaluable. Additionally, Barnabas' master students, Yuchen Shen and Chenhao Zhang, made meaningful contributions to the drug discovery project through their unique expertise.

I have also learned a lot about deep learning theory attending Andrej's group meeting with Bingbin Liu, Yuchen Li, Tanya Marwah, Ashwini Pokle, and Elan Rosenfeld. They provided me with a community as Barnabas went on sabbatical. I benefited from the support of Biswajit Paria, Manzil Zaheer, and Hai Pham, who are also Barnabas' students, and shared their perspectives and experiences during their PhD journeys. I am grateful for all the fun outings with Ezra Winston, Charvi Rastogi, Igor Gitman, Helen Zhou, Ian Char, Youngseog Chung, Paul Pu, Gabriele Farina, and many others. In addition, other senior students such as Anthony Platanios, Otilia Stretcu, Abulhair Saparov, Chris Dann, Mrinmaya Sachan, and Mariya Toneva have also provided me with guidance at the start of my PhD. Before focusing on generative models, I explored robotics in Manuela Veloso's lab, where I made friends with Anahita Mohseni-Kabir, Ashwin Khadke, Arpit Agarwal, Michiel De Jong, Kevin Zhang, Delvin Schwab, Vittorio Perera, and Traverse Rhodes.

I owe deep gratitude to Joelle Pineau for introducing me to the field of machine learning during my undergraduate studies at McGill University, and for continuing to mentor me after my graduation. Borja Balle, then a postdoc, met with me daily to explore time series data for robotics. I am also grateful to Stephanie Laflamme, Angus Leigh, Pierre-Luc Bacon, Gheorghe Comanici, and others from the Reasoning and Learning Lab for enriching my experience there.

On a personal level, my partner Robin Schmucker and my dog Maple have been steadfast pillars throughout my PhD journey. Maple's companionship, especially during our daily walks, kept me positive, active, and healthy. Robin's dedication to helping me proofread papers and his disciplined approach to life sharpened my research and communication skills. I also found solace and community in the Pittsburgh Chinese Church Oakland, particularly within the youth group, which became my home away from home. I am especially thankful to Songen and his wife Meiyu for generously hosting festival celebrations and welcoming everyone into their home.

Lastly, I owe immense gratitude to my father. Although he passed away when I was 17, the values and support he instilled in me have had a lasting impact. He was my greatest advocate, always encouraging me to aim higher and dream bigger. His decision to allow me to study abroad opened doors to opportunities I never imagined, and for that, I am forever thankful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Generative models learn the distribution of training data in order to generate novel data points from it. They have wide applications across many fields, such as computer vision [Goodfellow et al., 2014, Oord et al., 2016, 2017, Vahdat and Kautz, 2020], natural language processing [Devlin et al., 2018, Radford et al., 2019]. Some examples of generative models are generative adversarial network (GAN) [Goodfellow et al., 2014], variational autoencoders (VAE) [Kingma and Welling, 2013], diffusion models [Ho et al., 2020, Song et al., 2020a, Song and Ermon, 2019, Song et al., 2020b], autoregressive models, and normalizing flows [Rezende and Mohamed, 2015] etc. They transform some variables from a distribution, oftentimes Gaussian distribution, to a target distribution that the training dataset belongs to. [Goodfellow et al., 2016] states that many real-life data, such as images and texts, are supported on a low-dimensional manifold embedded in a high-dimensional ambient space. Recovering the low-dimensional manifold that the data lie on thus becomes an important step for learning the data distribution. Generative modeling can also relate to representation learning [Bengio et al., 2013] when a lower-dimensional latent representation is inferred in the process of learning the output data distribution, such as the VAEs.

Generative models have many well-known applications, such as images and languages. Particularly, conditional generation is a useful capability that has been widely adopted for commercial purposes. For example, image inpainting [Yu et al., 2018, Lugmayr et al., 2022] models are able to fill in the missing or corrupted pixels of images, language-conditioned image generation models such as Stable Diffusion [Rombach et al., 2022] and Dall-E [Ramesh et al., 2021] can generate images out of this world at a simple command, and language translation features are now ubiquitous in our digital life. Generative models have also brought disruption to applications such as program generation and drug discovery. Foundation models for code generation, such as Code Llama [Roziere et al., 2023] and AlpahCode [Li et al., 2022], are now capable of solving competitive-level coding problems, albeit unreliably. The field of drug discovery also sees immense progress when applying machine learning to drug discovery. AlphaFold [Jumper et al., 2021] can predict a protein's 3D structure from its amino acid sequences and was ranked #1 in the Critical Assessment of Structure Prediction (CASP) competition, significantly outperforming the second-place competitor. DiffDock [Corso et al., 2022] applies diffusion models to the problem of pose generation for protein docking, beating all previous approaches, including the previous state-of-the-art search-based approaches. A plethora of generative models' applications have truly brought technological disruption to our modern world.

Despite all the progress we have made in the past years, there are still many challenges to be tackled. For example, hallucination is a common problem among generative models, it manifests itself in the Large Language Model in the form of wrong answers to questions and, in drug discovery, it presents itself as molecules that do not exist in the natural world. Generative models nowadays are often very large in size with up to a trillion parameters, making them reliant on large quantities of good-quality data. This thesis will explore the mechanics of generative modeling, particularly in the case of VAE. Another focus of this thesis is to improve the generation of discrete data, with direct applications in molecule generation and program synthesis. Furthermore,

can we make use of the advancements and understanding in the continuous data generation to improve discrete data generation?

The application of generative models often involves generating novel data that fulfill certain objectives – to be similar in properties to the training data or achieve certain conditioned or unconditioned objectives. In this thesis, we want to dive deeper into methods that help generative models satisfy the objectives. The two underlying themes of this thesis are i) studying the deficiencies of the current generative models, how they affect generation quality, and ways to mitigate their effects; ii) how to improve data generation quality under computational constraints by leveraging domain knowledge.

Here, we provide a brief summary of the future chapters that will tackle

- In Chapter 2 (based on the paper Zhou et al. [2021]), we studied a program synthesis problem that touches on the challenges in discrete data generation. Given a geometric image as input, the goal of the project is to generate the context free grammar programs (CFG) that can be used to recreate the original image. In this work, we treat a non-differentiable renderer as a decoder and learn an interpretable encoder that leverages CFG grammar to generates a low-dimensional program representation of geometric images.
- Chapter 3 (based on the paper Koehler et al. [2021]) studies VAE's behavior in manifold and density recovery – for non-linear synthetic data, we found that VAE is not guaranteed to recover the underlying manifold that the high-dimensional training data lie on. We further studied the effects of a multi-stage VAE and showed it can improve manifold recovery on synthetic datasets.
- In Chapter 4 (based on the paper Zhou and Poczos [2023]), we apply the multi-stage VAE that was shown to improve manifold recovery of synthetic data to the task of molecule generation. We showed improvement in properties among the generated molecules on two tasks: a general generation task trained on ChEMBL dataset Mendez et al. [2019] and a generation task for protein targets fine-tuned on two curated smaller molecule sets.
- In Chapter 5, we incorporate a non-differentiable chemistry oracle that provides information on the net force of each atom in molecules and uses it to guide the reversed diffusion sampling process in order to achieve better stability among the final samples.

1.1 Generative Models

Generative models are probabilistic models that learn the distribution of a particular type of data, denoted as X for the rest of the thesis. If we know the distribution $P(X)$, we are able to sample novel data points from it. Oftentimes, we do not know the explicit distribution. Instead, we are given a dataset that includes individual points x's drawn from it and generative models learn a distribution that explains these data points. A variety types of generative models have been proposed, including autoregressive models, variational autoencoders, diffusion models, and generative adversarial models. We will show their applications in the later sections of this thesis. Here we include a brief overview of these different types of generative models.

Variational autoencoder (VAE) Kingma and Welling [2013] is one of the common generative models studied in this thesis. The variational inference framework assumes that the data x is generated from a latent variable $z \sim p(z)$. The prior $p(z)$ is assumed to be a multivariate standard normal distribution in the application of a VAE. A VAE model seeks to maximize the likelihood of the data, denoted as $\log p_\theta(x) = \log \int p(z)p_\theta(x|z)dz$ where θ denotes the generative parameters. However, the marginalization is intractable in practice due to the inherent complexity of the generator, or the decoder, thus an approximation of the objective is needed. Let x_i be an i.i.d. molecular sample from a discrete distribution. The VAE model assumes each of the output x_i is generated from a latent variable z follows the conditional distribution $p(x|z)$. However, in the situations where VAE is used, the posterior $p(z|x)$ is generally intractable. Let θ and ϕ denote the generative parameters and variational parameters respectively, the VAE model consists of a tractable encoder $q_\phi(z|x)$ and a decoder $p_\theta(x|z)$. Together, they approximate a lower bound to the log-likelihood of the data. By optimizing this lower bound we aim to increase the likelihood. This approximation enables the efficient posterior inference of the latent variable z given the output x_i and for marginal inference of the output variable x. The objective function of VAE is:

$$\mathcal{L}(\theta, \phi; x) = -D_{KL}(q_\phi(z|x) \,||\, p(z)) + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] \leq \log p_\theta(x) \tag{1.1}$$

For a generation, latent variable z_i is sampled from the prior $p(z)$ which is a multivariate standard normal and the decoder transforms z_i into the output x_i.

Many variations of VAEs are proposed, Most notable ones include Vector Quantized Variational Autoencoder (VQ-VAE) [van den Oord et al., 2017, Razavi et al., 2019b] and Deep Hierarchical Variational Autoencoder (NVAE) [Vahdat and Kautz, 2020]. VQ-VAE uses discrete latent representations and learns the prior. It has shown impressive results on image, video, and audio data. NVAE proposed a deep hierarchical VAE architecture with a new residual parameterization of the approximate posteriors and can obtain state-of-the-art results on several image datasets. In this thesis, we will discuss a Multi-Stage VAE which differs from NVAE in that multiple VAEs are trained sequentially, whereas NVAE, though it includes multiple latent layers, is trained as a single one. The training dynamics of the two models are thus entirely different. Most recently, VAE has also been combined with a diffusion model in stable diffusion [Rombach et al., 2022] to produce state-of-the-art performance.

Diffusion Model Current work on diffusion model consists of three main directions – Denoising Diffusion Probabilistic Models [Ho et al., 2020, Dhariwal and Nichol, 2021, Sohl-Dickstein et al., 2015, Song et al., 2020a] and Score-based Generative Model [Song and Ermon, 2019, Song et al., 2020b]. Here, we provide a more detailed introduction to DPPM because it is relevant to our work in Section 5. A DDPM consists of a forward *diffusion process* and a reverse *denoising process*. The diffusion process is a Markov chain that gradually adds Gaussian noises with a variance schedule $\beta_{1:T}$ from timestep 1 to T to the original datapoint $\boldsymbol{x}_0$. The schedule is chosen such that $\boldsymbol{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The forward diffusion process q is usually defined as a fixed schedule by the following:

$$q(\boldsymbol{x}_{1:T} \mid \boldsymbol{x}_0) = \prod_{t=1}^{T} q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}) \qquad q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \mathbf{I}) \tag{1.2}$$

where $\beta_{1:T}$ is pre-defined ahead of training. The reverse denoising process starts with $\boldsymbol{x}_T$ and recovers the original datapoint $\boldsymbol{x}_0$ by predicting the mean of $\boldsymbol{x}_{t-1}$ given $\boldsymbol{x}_t$, denoted as $\mu_\theta(\boldsymbol{x}_t, t)$ where θ is the parameter. We can model the reverse process as follows:

$$p_\theta(\boldsymbol{x}_{0:T}) = p(\boldsymbol{x}_T) \prod_{t=1}^{T} p_\theta(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t) \qquad p_\theta(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{x}_{t-1}; \mu_\theta(\boldsymbol{x}_t, t), \Sigma_\theta(\boldsymbol{x}_t, t)) \tag{1.3}$$

In practice, Σ_θ is set to be $\sigma_t^2 I$ for all t for simplicy, and $\sigma_t = \sqrt{1-\alpha_t^2}$ and $\alpha_t = \sqrt{\prod_{i=1}^{t}(1-\beta_i)}$. Ho et al. [2020] further simplified the objective from predicting mean $\mu_\theta(\boldsymbol{x}_t, t)$ to predict the noise at each step, which is denoted by $\epsilon_\theta(\boldsymbol{x}_t, t)$, as $\boldsymbol{x}_t = \alpha_t \boldsymbol{x}_0 + \sigma_t \epsilon$, and $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The training objective is minimize $\mathbb{E}_{\boldsymbol{x}_0, \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), t}\left[\|\epsilon - \epsilon_\theta(\boldsymbol{x}_t, t)\|^2\right]$, and the original $\mu_\theta(\boldsymbol{x}_t, t)$ can be parameterized as $\frac{1}{1-\beta_t}(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t^2}}\epsilon_\theta(\boldsymbol{x}_t, t))$. Consequently, we have

$$\boldsymbol{x}_{t-1} \sim \mathcal{N}(\frac{1}{1-\beta_t}(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t^2}}\epsilon_\theta(\boldsymbol{x}_t, t)), \sigma_t^2 \mathbf{I}) \tag{1.4}$$

To generate via the score-based diffusion model [Song and Ermon, 2019] (as opposed to DDPM we just described), the output is sampled via Langevin Dynamics using a score function $\nabla_x \log p(x)$:

$$\tilde{\boldsymbol{x}}_t = \tilde{\boldsymbol{x}}_{t-1} + \frac{\sigma}{2}\nabla_x \log p(\tilde{\boldsymbol{x}}_{t-1}) + \sqrt{\epsilon}\boldsymbol{z}_t \tag{1.5}$$

where $z_t \sim \mathcal{N}(0, I)$ and $\epsilon > 0$ is a fixed step size. $\tilde{\boldsymbol{x}}_0 \sim \pi(\boldsymbol{x}_0)$ where π is the prior. $\tilde{\boldsymbol{x}}_T$ equals $p(\boldsymbol{x})$ when $\epsilon \to 0$ and $T \to \infty$. The score-based generative model trained a score function $s_\theta(\tilde{\boldsymbol{x}})$ parameterized with θ to estimate the $\nabla_x \log p(\tilde{\boldsymbol{x}}_{t-1})$ term.

Generative Adversarial Network (GAN) [Goodfellow et al., 2014] GAN is a framework that consists of a generator $\mathcal{G}$ that learns the distribution of the training data and a discriminator $\mathcal{D}$ that classifies if a sample comes from the original data distribution. The discriminator and the generator plays a twp-player minimax game with the following value function:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})}[\log \mathcal{D}(\boldsymbol{x})] + \mathcal{E}_{z \sim p_{\boldsymbol{z}}(\boldsymbol{z})}(\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}))))) \tag{1.6}$$

where $\boldsymbol{z}$ generally follows a normal distribution as the initial noise to be transformed by the generator. We alternate between k-step of training the discriminator and one step of training the generator.

Arjovsky et al. [2017] proposed a variant of GAN called Wasserstein GAN that improves the stability of the minimax training and reduces mode collapse, which means that the generated samples only capture a single or a few modes of the training data as opposed to the full range. It introduces a critic function f parameterized by ω in place of the discriminator during training to help minimize the Wasserstein distance between the generator distribution and the data distribution. The critic does not play the same role as the discriminator and is trained according to the following loss:

$$\mathcal{L}_{\text{critic}}(\omega) = \max_{\omega \in W} \mathbb{E}_{\boldsymbol{x} \sim P_{data}}[f_\omega(\boldsymbol{x})] - \mathbb{E}_{z \sim P_z}[f_\omega(g(z))] \tag{1.7}$$

Naturally, the generator $g(\cdot)$ is trained by minimizing the same loss.

Besides other variants of loss functions, such as MMD GAN [Li et al., 2017], new architectures have also been proposed for GAN to scale up to higher-resolution images [Brock et al., 2018], to perform style transfer [Zhu et al., 2017]. Overall, the development of GAN over the years has made it a ubiquitous model for image generation.

Autoregressive Models Given past output values, autoregressive models work as a next token predictor. It models the distribution $p(\mathbf{x}_i | \mathbf{x}_1 = \boldsymbol{x}_1, \mathbf{x}_2 = \boldsymbol{x}_2, \cdots, \mathbf{x}_{i-1} = \boldsymbol{x}_{i-1})$ at each iteration. The probability of the overall output can be represented as the following:

$$P(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n) = \prod_{i=1}^{N} P(\mathbf{x}_i = \boldsymbol{x}_i | \mathbf{x}_1 = \boldsymbol{x}_1, \mathbf{x}_2 = \boldsymbol{x}_2, \cdots, \mathbf{x}_{i-1} = \boldsymbol{x}_{i-1}) \tag{1.8}$$

The neural network used to model such data has evolved over the years from RNN [Rumelhart et al., 1986] and LSTM [Hochreiter and Schmidhuber, 1997, Graves and Graves, 2012] to transformers [Vaswani et al., 2017]. Transformers can overcome some of the challenges of RNN and LSTM such as vanishing/exploding gradients due to the recurrent nature of the models and are better at capturing long-term dependencies. Common applications for autoregressive models have been sequential data such as language and time series data. Recent developments of large foundation models with up to a trillion parameters, such as GPT [Achiam et al., 2023] and Llamma [Touvron et al., 2023], have greatly boosted the profile of autoregressive models and become the state-of-the-art model for sequential data. They can also be "repurposed" to different tasks through fine-tuning. Beyond generating sequential data, transformer-based models for image generation such as ViT [Dosovitskiy et al., 2020] have shown comparable results on image benchmark datasets to the state-of-the-art convolutional neural network models. Each image is divided into ordered patches flattened before inputting to the model, positional embeddings are added to preserve the order information, similar to sequential data.

In the following sections, we will show how these generative models can be applied to tasks such as program synthesis and molecule generation.

Chapter 2

Learning Discrete Representation of Geometric Images

Image generation is extensively studied in machine learning and computer vision literature. Vast numbers of papers have explored image generation through low-dimensional latent representations [Goodfellow et al., 2014, Arjovsky et al., 2017, Li et al., 2017, Kingma and Welling, 2013, van den Oord et al., 2017, Oord et al., 2016]. However, it is challenging to learn disentangled representations that allow control over each component of the generative models separately [Higgins et al., 2017, Kim and Mnih, 2018, Locatello et al., 2018, Chen et al., 2016]. In this sectin of the thesis, we tackle the problem of CFG program generation from constructive solid geometry (CSG) [Hubbard, 1990] and computer aided design (CAD) images, which are commonplace in engineering and design applications. Parsing a geometric image into a CFG program not only enables selective manipulations of the desired components while preserving the rest, but also provides a human-readable alternative to the opaque low-dimensional representations generated by neural networks. Our model for extracting the programs can be viewed as an encoder and the renderer that reconstructs the image as a decoder. We focus on a non-differentiable renderer (most design software are non-differentiable, e.g. Blender and Autodesk). They are more common than differentiable ones [Li et al., 2018a, Liu et al., 2019, Kania et al., 2020], but they are also more challenging to work with neural networks due to their discrete nature not easily integrated into the network and the gradients w.r.t. the rendered images being *inaccessible*.

A common scheme for parsing images (e.g. CSG images) into programs (e.g. CFG programs) for non-differentiable renderers involves two steps: first use synthetic images with ground-truth programs for supervised pretraining, followed by REINFORCE fine-tuning [Sharma et al., 2018, Ellis et al., 2019] on the target image dataset. Sampling programs from a grammar can provide data suitable for supervision if the target images are restricted to combinations of geometric primitives specified in the grammar. There are two limitations of supervised methods. Firstly, it maximizes the likelihood of a single reference program while penalizing many other correct programs [Bunel et al., 2018] using maximum likelihood estimation (MLE). This observation is known as *program aliasing* and it adversely affects supervise learning's performance. Secondly, it does not generalize well to the test images not generated by the grammar. REINFORCE fine-tuning is proposed to remedy the two limitations [Bunel et al., 2018, Sharma et al., 2018]. The transition between the supervised pretraining and REINFORCE fine-tuning is delicate, however,

because the model is sensitive to bad gradient updates that cause the grammar structure to collapse among the generated programs. Additionally, the quality of the curated pretraining dataset can limit the downstream model's ability to generalize. In this thesis, however, we focus on the more interesting and more challenging *unsupervised task*, when ground-truth programs of the images are not available for training.

Because of the problem's discrete nature, we rely on tools from reinforcement learning, such as REINFORCE. However, the program space grows exponentially with the length of the program and valid programs are too sparse in the search space to be sampled frequently enough to learn. Training with the naive REINFORCE provides no performance gain in our experiments. RL techniques such as Hindsight Experience Replay [Andrychowicz et al.] that mitigate the sparse reward problem cannot be applied here due to the Markov assumption on the model. We improve the sample efficiency of the REINFORCE algorithm and show that our improved approach achieves competitive results to a two-step model. We further demonstrate that our method generalizes better on a synthetic 2D CSG dataset than a supervised method. On a 2D CAD dataset, which was *NOT* generated by CFG and thus cannot be captured sufficiently by a synthetic dataset, our method exceeds the results of a pretrained model by a large margin and performs competitively to the refined models.

Here we summarize the key ingredients that help successfully learn to parse an image *without program supervision* while using a non-differentiable renderer *without direct gradient propagation* w.r.t. the rendered images:

- We incorporate a **grammar-encoded tree LSTM** to impose a structure on the search space such that the algorithm is sampling a path in the CFG syntax tree top-down. This guarantees validity of the output program.
- We propose an **entropy estimator** suitable for sampling top-down from a syntax tree to encourage exploration of the search space by entropy regularization.
- Instead of relying on naive Monte Carlo sampling, we adopt **sampling without replacement** from the syntax tree to obtain better entropy estimates and REINFORCE objective for faster convergence.

2.1 Related Work

Research on converting images to programs relates more closely to our work [Sharma et al., 2018, Ellis et al., 2019, 2018, Liu et al., 2018b, Shin et al., 2019, Beltramelli, 2018, Kania et al., 2020]. Tian et al. [2019], Kania et al. [2020] incorporate differentiable renderers into the learning pipeline while we treat our renderer as an external process independent from the learning process, thus unable to propagate gradient through the renderer. Furthermore, Kania et al. [2020] construct the parse tree bottom up and pre-determines the number of leaves, as opposed to top down like ours which is more general. Ellis et al. [2018] use neural networks to extract shapes from hand-drawn sketches, formulate the grammatical rules as constraints and obtain the final programs by optimizing a constraint satisfaction problem. Similarly, Du et al. [2018] cast the problem of parsing a 3D model into a CSG tree as a constraint satisfaction problem and solve by an existing SAT solver. This process can be computationally expensive compared to neural network

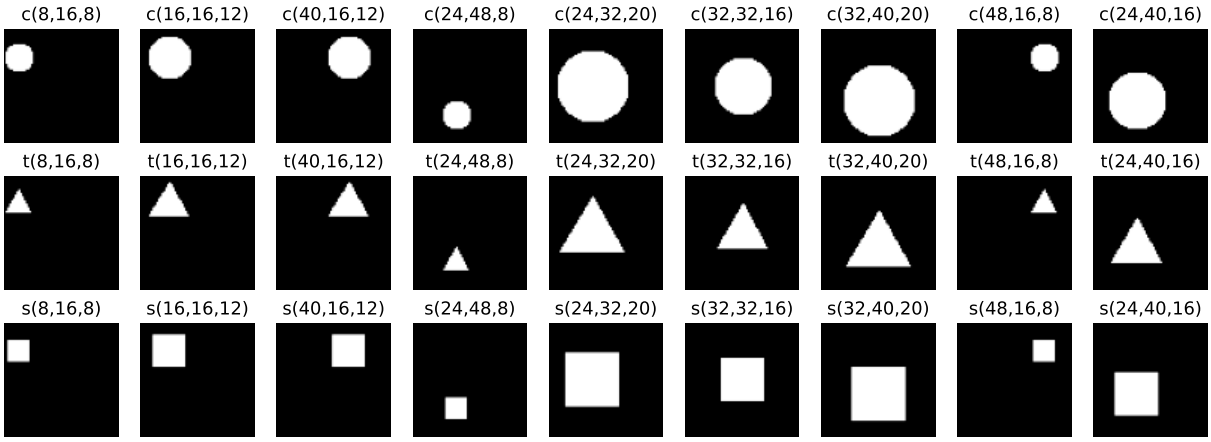


Figure 2.1: Each shape encoding is on top of the image it represents.

based solutions. More relevantly, Sharma et al. [2018] perform program synthesis by supervised pretraining before RL fine-tuning to generalize to a CAD dataset. Ellis et al. [2019] pretrain a policy with supervision from synthetic data and learns a value function with REINFORCE. Both are used for pruning unpromising candidates during test time. Their reward function is binary and cannot approximate images not generated by a grammar, as opposed to our model.

2.2 Proposed Algorithm

CSG Image and CFG Program. We use constructive solid geometry (CSG) [Hubbard, 1990] to describe an image. The input of our model are images constructed from geometric shapes (e.g. square, circle, ...) each with a designated size and location (see Figure 2.1). The outputs of the model are context-free grammar (CFG) programs. In the CFG specification [Sharma et al., 2018], S, T, and P are non-terminals for the start, operations, and shapes. The rest are terminals, e.g. $+$ (union), $*$ (intersection), $-$ (subtraction), and $c(48, 16, 8)$ stands for a circle with radius 8 at location $(48, 16)$ in the image. Figure 2.5 contains examples of CSG images and their corresponding programs. Each line below is a *production rule* or just *rule* for simplicity:

$$S \to E \tag{2.1}$$
$$E \to EET|P \tag{2.2}$$
$$T \to +|-|* \tag{2.3}$$
$$P \to SHAPE_1|SHAPE_2|\cdots|SHAPE_n. \tag{2.4}$$

2.2.1 Learning with REINFORCE

A model trained with the REINFORCE objective only [Sharma et al., 2018] is unable to improve beyond the lowest reward (Section 2.3.1). We propose three components to enable the RL approach to learn in this setting: (i) a tree model entropy estimator to encourage exploration;

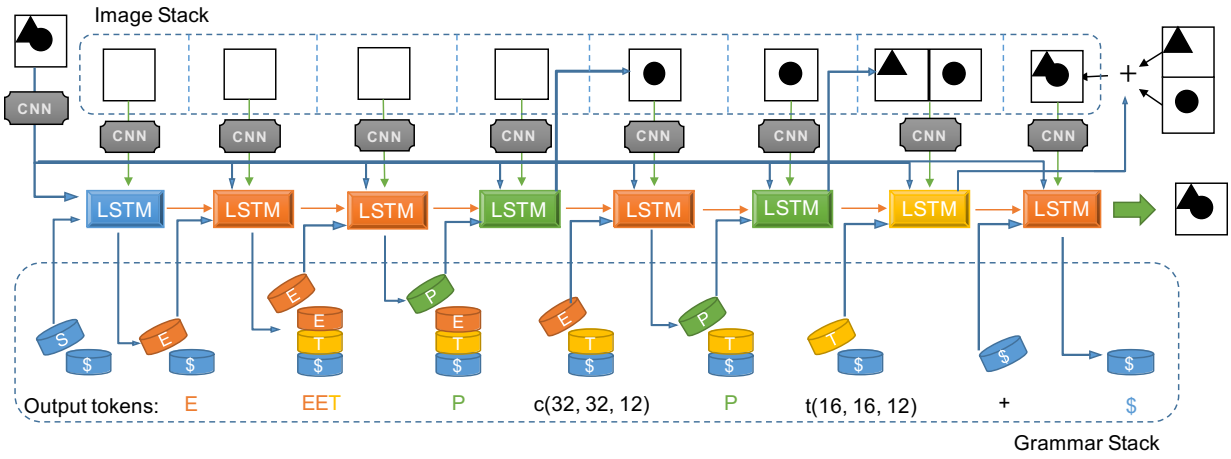


Figure 2.2: This is an example of the grammar encoded tree LSTM at work. The top layer of images demonstrates the image stack and the bottom layer demonstrates the grammar stack. The blue, orange, yellow and green colored LSTM cell generates grammatical tokens according to the CFG rule 2.1, 2.2, 2.3 and 2.4 respectively. In implementation, we can constrain the output space by adding a mask to the output of the LSTM and render the invalid options with close to zero probability of being sampled.

(ii) sampling without replacement in the program space to facilitate optimization and further encourage exploration; (iii) a grammar-encoded tree LSTM to ensure valid output sequences with an image stack to provide intermediate feedback. We start this section by discussing objective and reward function.

Objective Function Our model consists of a CNN encoder for input images, an embedding layer for the actions, and an RNN for generating the program sequences (see Figure 2.2). The model is trained with entropy regularized REINFORCE [Williams, 1992]. Here, let $\mathcal{H}(s)$ and $f(s)$ stand for the entropy (we will define this later) and reward function of sequence s, respectively, and let θ denote the parameters of the model. The objective is optimized as follows

$$\Delta\theta \propto \mathbb{E}_{s\sim P_\theta(s)}[\nabla_\theta \log P(s) f(s)] + \alpha \nabla_\theta \mathcal{H}(s) \tag{2.5}$$

Reward Function The output program s is converted to an image $\mathbf{y}$ by a non-differentiable renderer. The image is compared to the target image $\mathbf{x}$ and receives a reward $f(s) = R(\mathbf{x}, \mathbf{y})$. We use Chamfer Distance (CD) as part of the reward function. The CD calculates the average matching distance to the nearest feature and is a greedy estimation of image similarity, unlike Optimal Transport (OT). However, OT is not computationally feasible for RL purpose.

Formally, let $x \in \mathbf{x}$ and $y \in \mathbf{y}$ be pixels in each image respectively. Then the distance $Ch(\mathbf{x}, \mathbf{y})$ is

$$Ch(\mathbf{x}, \mathbf{y}) = \frac{1}{|\mathbf{x}|} \sum_{x\in\mathbf{x}} \min_{y\in\mathbf{y}} ||x - y||_2 + \frac{1}{\|\mathbf{y}\|} \sum_{y\in\mathbf{y}} \min_{x\in\mathbf{x}} ||x - y||_2 \tag{2.6}$$

The CD is scaled by the length of the image diagonal (ρ) [Sharma et al., 2018] such that the final value is between 0 and 1. For this problem, the reward $1 - Ch(\mathbf{x}, \mathbf{y})$ mostly falls between 0.9 and 1. We exponentiate $1 - Ch(\mathbf{x}, \mathbf{y})$ to the power of $\gamma = 20$ to achieve smoother gradients

[Laud, 2004]. We add another pixel intersection based component to differentiate shapes with similar sizes and locations. The final reward function is defined as:

$$R(\mathbf{x}, \mathbf{y}) = \max(\delta, (1 - \frac{Ch(\mathbf{x}, \mathbf{y})}{\rho})^{\gamma} + \frac{\sum_{x \in \mathbf{x} \cap \mathbf{y}} 1}{\sum_{x \in \mathbf{x}} 1}) \tag{2.7}$$

The first and second part of the reward function provide feedback on the physical distance and similarity of the prediction respectively. We clip the reward below $\delta = 0.3$ to simplify it when the quality of the generated images are poor. A low reward value provides little insight on its performance and is largely dependent on its target image. Similar reward clipping idea was used in DQN [Mnih et al., 2013].

Algorithm 1 Sampling w/o Replacement Tree LSTM

Input: Target Image $\mathbf{x}$, Number of samples k

Initialize: Grammar stack S, Image stack I, Sample set $\mathbb{B}$

Encode the target image $\tilde{T} \leftarrow \text{Encode}(T)$

$\mathbb{B} = \{\mathbf{s}_0^i, G_{s_0^i}, \phi_{i,0} | \mathbf{s}_0^i = \emptyset, G_{s_0^i} = 0, \phi_{i,0} = 0\}$

for $i \in 1, 2, \cdots k+1$ and $\mathcal{H}(v) = 0$

for $j := 1$ to n **do**

$\mathbf{\Phi}_{:,j}, \mathcal{H}_{i,j} \leftarrow \text{TreeLSTM}(S, I, \mathbb{B}, \mathbf{x})$ (Algorithm 3)

$\mathbb{B} \leftarrow \text{Sample\_w/o\_Replacement}(\mathbf{\Phi}_{:,j}, \mathbb{B}, k)$ (See Algorithm 2 and [Kool et al., 2019b])

$\hat{\mathcal{H}}_D \leftarrow \hat{\mathcal{H}}_D + \sum_{j=1}^{n} \frac{1}{W_j(S)} \sum_{s^i \in S} \frac{p_\theta(s_j^i)}{q_{\theta,\kappa}(s_j^i)} \mathcal{H}_{i,j}$ (Equation 2.16)

if $s_j^i \in \mathcal{G}$ **then** $S_i.\text{push}(s_j^i)$ **else** $I_i.\text{push}(s_j^i), \forall s^i \in \mathbb{B}$

end for

$\mathbf{y}_i = \text{Render}(s^i)$ for $i \in 1, 2, \cdots k$

Maximize $\mathbb{E}[\sum_{i=1}^{k} R(\mathbf{x}, \mathbf{y}_i)] + \alpha \hat{\mathcal{H}}_D$

2.2.2 Exploration with Entropy Regularization

Entropy regularization in RL is a standard practice for encouraging exploration. Here we propose an entropy estimation for sampling top-down from a syntax tree. Let S denote a random variable of possible programs. Its entropy is defined by $\mathcal{H}(S) = \mathbb{E}[-\log P(S)]$ [1]. Estimating $\mathcal{H}(S)$ is challenging because the possible outcomes of S is exponentially large and we cannot enumerate all of them. Given the distribution P, the naive entropy estimator is

$$\hat{\mathcal{H}} = -\frac{1}{K} \sum_{i=1}^{K} \log P(s^i), \tag{2.8}$$

where $\{s^i\}_{i=1}^K$ are iid samples from P. In practice, when K is not exponentially large, this estimator has huge variance. We provide an improved estimator designed for our syntax tree: First, we decompose the program S into $S = X_1 \dots X_n$, where each X_j is the random variable

[1] Following Rule equation 2.1 and Rule equation 2.2 we overload S and P.

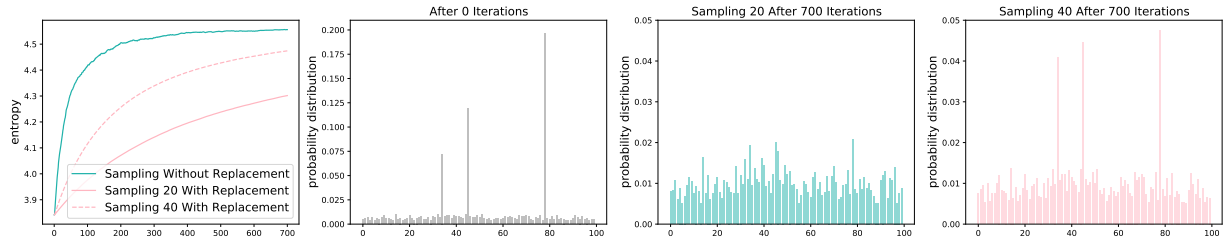


Figure 2.3: The left most image demonstrates the entropy value increases over 700 iterations by sampling 20 distinct samples with and without replacement as well as sampling 40 samples with replacement. The second image shows the initial distribution. The third and fourth images show the final distributions.

for the token at position j in the program. Under autoregressive models (e.g. RNN), we can access the conditional probabilities, and this allows us to construct decomposed entropy estimator $\hat{\mathcal{H}}_D$ as

$$\hat{\mathcal{H}}_D = \frac{1}{K}\sum_{i=1}^{K}\sum_{j=1}^{n}\mathcal{H}(X_j|X_1 = x_1^i, \cdots, X_{j-1} = x_{j-1}^i), \tag{2.9}$$

where $s^i = x_1^i, \ldots, x_n^i$, and $\mathcal{H}(X_j|X_1 = x_1^i, \cdots, X_{j-1} = x_{j-1}^i)$ is the conditional entropy. The below lemma states that $\hat{\mathcal{H}}_D$ is indeed an improved estimator over $\hat{\mathcal{H}}$.

Lemma 2.2.1. *The proposed decomposed entropy estimator $\hat{\mathcal{H}}_D$ is unbiased with lower variance, that is $\mathbb{E}[\hat{\mathcal{H}}_D] = \mathcal{H}(S)$ and $Var(\hat{\mathcal{H}}_D) \leq Var(\hat{\mathcal{H}})$.*

Proof. Entropy of a sequence can be decomposed into the sum of the conditional entropy at each step conditioned on the previous values. This is also called the chain rule for entropy calculation. Let $X_1, X_2, \cdots, X_n$ be drawn from $P(X_1, X_2, \cdots, X_n)$ [Cover and Thomas, 2012]:

$$\mathcal{H}(X_1, X_2, \cdots, X_n) = \sum_{j=1}^{n}\mathcal{H}(X_j|X_1, \cdots, X_{j-1}) \tag{2.10}$$

If we sum up the empirical entropy at each step after the softmax output, we can obtain an unbiased estimator of the entropy. Let S be the set of sequences that we sampled and each sampled sequence s^i consists of $X_1, X_2, \cdots, X_n$:

$$\begin{aligned}
&\mathbb{E}_{X_1,\ldots,X_{j-1}}(\hat{\mathcal{H}}_D) \\
&= \mathbb{E}_{X_1,\ldots,X_{j-1}}(\frac{1}{|S|}\sum_{i\in|S|}\sum_{j=1}^{n}\mathcal{H}(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) \\
&= \frac{1}{|S|}\cdot|S|\sum_{j=1}^{n}\mathcal{H}(X_j|X_1, \cdots, \quad X_{j-1}) \\
&= \mathcal{H}(X_1, X_2, \cdots, X_n)
\end{aligned}$$

In order to incorporate the stepwise estimation of the entropy into the beam search, we use the similar reweighting scheme as the REINFORCE objective. The difference is that the REINFORCE

objective is reweighted after obtaining the full sequence because we only receive the reward at the end and here we reweight the entropy at each step. We denote each time step by j and each sequence by i, the set of sequences selected at time step j is S_j and the complete set of all possible sequences of length j is T_j and $S_j \in T_j$. We are taking the expectation of the estimator over the $G_{\phi_{i,j}}$ scores. As we discussed before, at each step, each potential beam receives a stochastic score $G_{\phi_{i,j}}$. The beams associated with the top-$k+1$ stochastic scores are chosen to be expanded further and κ is the $k+1$-th largest $G_{\phi_{i,j}}$. κ can also be seen as a threshold in the branching selection process and $q_{\theta,\kappa}(s_j^i) = P(G_{s_j^i} > \kappa) = 1 - \exp(-\exp(\phi_{i,j} - \kappa))$. For details on the numerical stable implementation of $q_{\theta,\kappa}(s_j^i)$, please refer to [Kool et al., 2019b].

$$
\begin{aligned}
&\mathbb{E}_{G_\phi}(\sum_{j=1}^{n}\sum_{s_j^i \in S_j}\frac{p_\theta(s_j^i)}{q_{\theta,\kappa}(s_j^i)}\mathcal{H}(X_j|X_1 = x_1^i, X_2 = x_2^i, \cdots, X_{j-1} = x_{j-1}^i)) \\
&= \sum_{j=1}^{n}\mathbb{E}_{G_\phi}(\sum_{i\in|T_j|}\frac{p_\theta(s_j^i)}{q_{\theta,\kappa}(s_j^i)}\mathcal{H}(X_j|X_1 = x_1^i, X_2 = x_2^i, \cdots, X_{j-1} = x_{j-1}^i))\mathbb{1}_{\{x_1^i,\cdots,x_j^i\}\in S_j}) \\
&= \sum_{j=1}^{n}\sum_{i\in|T_j|}p_\theta(s_j^i)\mathcal{H}(X_j|X_1 = x_1^i, X_2 = x_2^i, \cdots, X_{j-1} = x_{j-1}^i)\mathbb{E}_{G_\phi}(\frac{\mathbb{1}_{\{s_j^i = x_1^i,\cdots,x_j^i\}\in S_j}}{q_{\theta,\kappa}(s_j^i)}) \\
&= \sum_{j=1}^{n}\mathcal{H}(X_j|X_1, X_2, \cdots, X_{j-1})\cdot 1 \\
&= \mathcal{H}(X_1, X_2, \cdots, X_n)
\end{aligned}
$$

For the proof of $\mathbb{E}_{G_\phi}(\frac{\mathbb{1}_{\{s_j^i\in S_j}}{q_{\theta,\kappa}(s_j^i)}) = 1$, please refer to [Kool et al., 2019b], appendix D.

We abuse $\mathbb{E}_{X_j}$ to be $\mathbb{E}_{X_j|X_1,\ldots,X_{j-1}}$ and Var_{X_j} to be $\mathrm{Var}_{X_j|X_1,\ldots,X_{j-1}}$ to simplify the notations.

$$
\begin{aligned}
&\mathrm{Var}_{X_1,X_2,\cdots,X_n}(\frac{1}{|S|}\sum_{i\in|S|}\sum_{j=1}^{n}\mathcal{H}(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) \\
&= \frac{1}{|S|^2}\sum_{i\in|S|}\sum_{j=1}^{n}\mathrm{Var}_{X_1,X_2,\cdots,X_n}(\mathcal{H}(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) \\
&= \frac{1}{|S|^2}\sum_{i\in|S|}\sum_{j=1}^{n}(\mathbb{E}(\mathcal{H}^2(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) - \mathbb{E}^2(\mathcal{H}(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i))) \\
&= \frac{1}{|S|^2}\sum_{i\in|S|}\sum_{j=1}^{n}(\mathbb{E}_{X_1,\cdots,X_{j-1}}\mathbb{E}_{X_j}^2(\log P(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) \\
&\quad - \mathbb{E}_{X_1,\cdots,X_{j-1}}^2\mathbb{E}_{X_j}(\log P(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i))) \\
&= \frac{1}{|S|^2}\sum_{s^i\in S}\sum_{j=1}^{n}(\mathbb{E}_{X_1,\cdots,X_{j-1}}(\mathbb{E}_{X_j}(\log^2 P(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)) \\
&\quad - \mathrm{Var}_{X_j}(\log P(X_j|X_1 = x_1^i, \ldots, X_{j-1} = x_{j-1}^i)))
\end{aligned}
$$

$$
\begin{aligned}
&- \mathbb{E}^2_{X_1,\cdots,X_{j-1}} \mathbb{E}_{X_j}(\log P(X_j|X_1 = x^i_1, \ldots, X_{j-1} = x^i_{j-1})))\\
&= \frac{1}{|S|^2} \sum_{i\in|S|} \sum_{j=1}^{n} (\mathbb{E}_{X_1,\cdots,X_j}(\log^2 P(X_j|X_1 = x^i_1, \ldots, X_{j-1} = x^i_{j-1}))\\
&\quad - \mathbb{E}^2_{X_1,\cdots,X_j}(\log P(X_j|X_1 = x^i_1, \ldots, X_{j-1} = x^i_{j-1})))\\
&\quad - \mathbb{E}_{X_1,\cdots,X_{j-1}} \mathrm{Var}_{X_j}(\log P(X_j|X_1 = x^i_1, \ldots, X_{j-1} = x^i_{j-1}))\\
&= \frac{1}{|S|^2} \sum_{i\in|S|} \sum_{j=1}^{n} (\mathrm{Var}_{X_1,\cdots,X_j}(\log P(X_j|X_1 = x^i_1, \ldots, X_{j-1} = x^i_{j-1}))\\
&\quad - \mathbb{E}_{X_1,\cdots,X_{j-1}} \mathrm{Var}_{X_j}(\log P(X_j|X_1 = x^i_1, \ldots, X_{j-1} = x^i_{j-1}))\\
&\leq \frac{1}{|S|^2} \sum_{i\in|S|} \sum_{j=1}^{n} \mathrm{Var}_{X_1,\cdots,X_j}(\log P(X_j|X_1 = x^i_1, \ldots, X_{j-1} = x^i_{j-1}))\\
&= \mathrm{Var}_{X_1,X_2,\cdots,X_n}(\frac{1}{|S|} \sum_{i\in|S|} \log P(s^i))
\end{aligned}
$$

The fifth equation from the low variance proof holds from the fact that $\mathbb{E}^2_X \mathbb{E}_{Y|X}[f(X,Y)] = \mathbb{E}^2_{X,Y}[f(X,Y)]$. The result still stands after applying reweighting for the beam search. □

2.2.3 Effective Optimization By Sampling Without Replacement

After establishing our REINFORCE method with entropy regularization objective, now we show the intuition behind choosing sampling without replacement (SWOR) over sampling with replacement (SWR). For this explanation, we use a synthetic example (Figure 2.3).

We initialize a distribution of $m = 100$ variables with three of them having significantly higher probability than the others (Figure 2.3 (2)). The loss function is entropy. Its estimator is $\frac{1}{m}\sum_{i=1}^{m} \log p_i$ for SWR and $\sum_{i=1}^{m} \frac{p_i}{q_i} \log p_i$ for SWOR. In both cases, p_i is the i-th variable's probability. q_i is the re-normalized probability after SWOR. $\frac{p_i}{q_i}$ is the importance weighting. The increase in entropy by sampling 20 variables without replacement is more rapid than 40 variables with replacement. At the end of the 700 iterations, the distribution under SWOR is visibly more uniform than the other. SWOR would achieve better exploration than SWR.

To apply SWOR to our objective, the REINFORCE objective and the entropy estimator require importance weightings. Let s^i_j denotes the first j elements of sequence s^i:

$$
\nabla_\theta \mathbb{E}_{s\sim p_\theta(s)}[f(s)] \approx \sum_{s^i \in S} \frac{\nabla_\theta p_\theta(s^i)}{q_\theta(s^i)} f(s^i) \quad \text{and,} \tag{2.11}
$$

$$
\hat{\mathcal{H}}_D \approx \sum_{j=1}^{n} \sum_{s^i \in S} \frac{p_\theta(s^i_j)}{q_\theta(s^i_j)} \mathcal{H}(X_j|X_1 = x^i_1, \cdots, X_{j-1} = x^i_{j-1}) \tag{2.12}
$$

Implementing SWOR on a tree structure to obtain the appropriate set of programs S is challenging. It is not practical to instantiate all paths and perform SWOR bottom-up. Instead, we

adopt a form of stochastic beam search by combining top-down SWOR with Gumbel trick that is equivalent to SWOR bottom-up [Kool et al., 2019b]. The sampling process is summarized in Algorithm 2 and described below.

Algorithm 2 Sampling_w/o_Replacement

Input: Log probability at time j $\mathbf{\Phi}_{:,j}$ Beam Set $\mathbb{B}$,
Number of beams k

function SAMPLING_W/O_REPLACEMENT($\mathbf{\Phi}_{:,j}, \mathbb{B}, k$)
$\tilde{\boldsymbol{G}}_j \leftarrow \emptyset$
for $\mathbf{s}^i_{j-1}, G_{s^i_{j-1}} \in \mathbb{B}$ and $\vec{\boldsymbol{\phi}}_{i,j} \in \mathbf{\Phi}_{:,j}$ **do**
$\vec{\mathbf{G}}_{\boldsymbol{\phi}_{i,j}} \sim$ Gumbel$(\vec{\boldsymbol{\phi}}_{i,j})$
$Z_{i,j} \leftarrow \max(\vec{\mathbf{G}}_{\boldsymbol{\phi}_{i,j}})$
Aggregate the values in the vector $\tilde{\mathbf{G}}_{\phi_{i,j}}$
$\tilde{\boldsymbol{G}}_j \leftarrow \tilde{\boldsymbol{G}}_j \cup \tilde{\mathbf{G}}_{\phi_{i,j}}$
end for
Choose top $k+1$ values in $\tilde{\boldsymbol{G}}_j \in \mathbb{R}^{(k+1)\cdot A}$ and form the new beam set
$\tilde{\mathbb{B}} = \{\mathbf{s}^i_j, G_{s^i_j}, \phi^i_j | \mathbf{s}^i_j = \tilde{\mathbf{s}}^i_{j-1} \cup \tilde{s}^i_j, G_{s^i_j} = \tilde{G}_{i,j}, \phi_{i,j} = \log P(\mathbf{s}^i_j)\}$ where $i \in 1, 2, \cdots, k+1$
return $\tilde{\mathbb{B}}$
end function

At each step of generation, the algorithm chooses the top-$k+1$ beams to expand based on the $\vec{\mathbf{G}}_{\boldsymbol{\phi}_{i,j}}$ score at time step j in order to find the top-k stochastic sequences at the end. The use for the additional beam will be explained later. Let A be all possible actions at time step j, $\vec{\boldsymbol{\phi}}_{i,j} \in \mathbb{R}^A$ is the log probability of each outcomes of sequence i at time j plus the log probability of the previous $j-1$ actions.

$$\vec{\boldsymbol{\phi}}_{i,j} = \big[\log P(a_1), \log P(a_2), \ldots, \log P(a_A)\big] + \phi_{i,j-1} \cdot \vec{\mathbb{1}} \tag{2.13}$$

For each beam, we sample a Gumbel random variable $G_{\phi_{i,j,a}} = \text{Gumbel}(\phi_{i,j,a})$ for each of the element a of the vector $\vec{\boldsymbol{\phi}}_{i,j}$. Then we need to adjust the Gumbel random variable by conditioning on its parent's adjusted stochastic score $G_{s^i_{j-1}}$ being the largest (Equation 2.14) in relation to all the descendant elements in $\vec{\mathbf{G}}_{\phi_{i,j}}$, the resulting value $\vec{\tilde{\mathbf{G}}}_{\phi_{i,j}} \in \mathbb{R}^A$ is the adjusted stochastic score for each of the potential expansions.

$$\vec{\tilde{\mathbf{G}}}_{\phi_{i,j}} = -\log(\exp\left(-\vec{\mathbb{1}} \cdot G_{s^i_{j-1}}\right) - \exp\left(-\vec{\mathbb{1}} \cdot Z_{i,j}\right) + \exp\left(-\vec{\mathbf{G}}_{\boldsymbol{\phi}_{i,j}}\right)) \tag{2.14}$$

Here $Z_{i,j}$ is the largest value in the vector $\vec{\mathbf{G}}_{\boldsymbol{\phi}_{i,j}} = [G_{\phi_{i,j,a_1}}, G_{\phi_{i,j,a_2}}, \cdots, G_{\phi_{i,j,a_A}}]$ and $G_{s^i_{j-1}}$ is the adjusted stochastic score of i-th beam at step $j-1$ from the last iteration. Conditioning on the parent stochastic score being the largest in this top-down sampling scheme makes sure that

each leaf's final stochastic score G_{s^i} is independent, equivalent to sampling the sequences bottom up without replacement [Kool et al., 2019b].

Once we have aggregated all the adjusted stochastic scores in $\vec{\tilde{\mathbf{G}}}_{\phi_{i,j}}$ from all previous $k+1$ beams, we select the top-$k+1$ scored beams from $(k+1)*A$ scores for expansions. The selected $k+1$ adjusted stochastic scores become the $G_{s_j^i}$ $\forall i$ in the new iteration. Note that the reason that we maintain one more beam than we intended to expand is that we need the $k+1$ largest stochastic score to be the threshold during the estimation of the entropy and REINFORCE objective. This is explained next. Please refer to Algorithm 2 for details on the branching process.

The sampling without replacement algorithm requires importance weighting of the objective functions to ensure unbiasness. The weighting term is $\frac{p_\theta(s^i)}{q_{\theta,\kappa}(s^i)}$. $p_\theta(s^i)$ represents the probability of the sequence s^i (s^i is the i-th completed sequence and $p_\theta(s^i) = \exp \phi_i$) and S represents the set of all sampled sequences s^i for $i = 1, 2, \cdots, k$. $q_{\theta,\kappa}(s_j^i) = P(G_{s_j^i} > \kappa) = 1 - \exp(-\exp(\phi_{i,j} - \kappa))$, where κ is the $(k+1)$-th largest $G_{s_j^i}$ score for all i and $\phi_{i,j} = \log P(a_{t_1}, a_{t_2}, \cdots, a_{t_j})$ is the log likelihood of the first j actions of i-th sequence, can be calculated based on the CDF of the Gumbel distribution. It acts as a threshold for branching selection. We can use the log probability of the sequence here to calculate the CDF because the adjust stochastic scores $G_{s_j^i}$'s are equivalent to the Gumbel scores of sequences sampled without replacement from bottom up. During implementation, we need to keep an extra beam, thus $k+1$ beams in total, to accurately estimate κ in order to ensure the unbiasness of the estimator.

To reduce variance of our objective function, we introduce additional normalization terms as well as a baseline. However, the objective function is biased with these terms. The normalization terms are $W(S) = \sum_{s^i \in S} \frac{p_\theta(s^i)}{q_{\theta,\kappa}(s^i)}$ and $W^i(S) = W(S) - \frac{p_\theta(s^i)}{q_{\theta,\kappa}(s^i)} + p_\theta(s^i)$.

Incorporating a baseline into the REINFORCE objective is a standard practice. A baseline term is defined as $B(S) = \sum_{s^i \in S} \frac{p_\theta(s^i)}{q_{\theta,\kappa}(s^i)} f(s^i)$ and $f(s^i)$ should be the reward of the complete i-th program s^i in this case.

To put everything together, the exact objective is as follows [Kool et al., 2019a]:

$$\nabla_\theta \mathbb{E}_{s \sim p_\theta(s)}[f(s)] \approx \sum_{s^i \in S} \frac{1}{W^i(S)} \cdot \frac{\nabla_\theta p_\theta(s^i)}{q_{\theta,\kappa}(s_n^i)} (f(s^i) - \frac{B(S)}{W(S)}) \tag{2.15}$$

Entropy estimation uses a similar scaling scheme as the REINFORCE objective:

$$\hat{\mathcal{H}}_D(X_1, X_2, X_3, \cdots, X_n) \approx \sum_{j=1}^{n} \frac{1}{W_j(S)} \sum_{s^i \in S} \frac{p_\theta(s_j^i)}{q_{\theta,\kappa}(s_j^i)} \mathcal{H}(X_j | X_1 = x_1^i, \cdots, X_{j-1} = x_{j-1}^i) \tag{2.16}$$

where $W_j(S) = \sum_{s^i \in S} \frac{p_\theta(s_j^i)}{q_{\theta,\kappa}(s_j^i)}$ and s_j^i denotes the first j elements of the sequence s^i. The estimator is unbiased excluding the $\frac{1}{W_j(S)}$ term.

2.2.4 Grammar Encoded Tree LSTM

We introduce a *grammar-encoded tree LSTM* which encodes the production rules into the model, thus guaranteed to generate correct programs, and significantly reduce the search space during

the training [Kusner et al., 2017, Alvarez-Melis and Jaakkola, 2016, Parisotto et al., 2016, Yin and Neubig, 2017]. There are 3 types of production rules in the grammatical program generation – shape selection (P), operation selection (T), and grammar selection (E). Grammar selection in this problem setting includes $E \rightarrow EET$, and $E \rightarrow P$ and they decide whether the program would expand. We denote the set of shape, operations and non-terminal outcomes (e.g. EET in Rule equation 2.2) to be $\mathcal{P}$, $\mathcal{T}$ and $\mathcal{G}$ respectively. A naive parameterization is to let the candidate set of the LSTM output to be $\{S, \$\} \cup \mathcal{T} \cup \mathcal{P}$, where $\$$ is the end token, and treat it as a standard language model to generate the program [Sharma et al., 2018]. The model does not explicitly encode grammar structures, and expect the model to capture it implicitly during the learning process. The drawback is that the occurrence of valid programs is sparse during sampling and it can prolong the training process significantly.

The proposed model can be described as an RNN model with a *masking mechanism* by maintaining a grammar stack to rule out invalid outputs. We increase the size of the total output space from $2+|\mathcal{P}|+|\mathcal{T}|$ of the previous approach (e.g. [Sharma et al., 2018]) to $2+|\mathcal{P}|+|\mathcal{T}|+|\mathcal{G}|$ by including the non-terminals. During the generation, we maintain a stack to trace the current production rule. Based on the current non-terminal and its corresponding expansion rules, we use the masking mechanism to weed out the invalid output candidates. Take the non-terminal T for example, we mask the invalid outputs to reduce the candidate size from $2 + |\mathcal{P}| + |\mathcal{T}| + |\mathcal{G}|$ to $|\mathcal{T}|$ only. In this process, the model will produce a sequence of tokens, including grammatical, shape and operation tokens. We only keep the terminals as the final output program and discard the rest. The resulting programs are ensured to be grammatically correct. During the generation process, grammatical tokens are pushed onto the grammar stack while intermediate images and operations are pushed onto an *image stack*. Images in the image stack are part of the input to the LSTM to aid the inference in the search space. Figure 2.2 is a visual representation of the process.

We provide a guide for the tree-LSTM illustration in Figure 2.2. This guide follows the arrows in the illustration (Figure 2.2) from left to right:

- A grammar stack with a start token S and an end token $\$$ as well as an empty image stack is initialized.
- In the first iteration, the token S is popped out. Following Rule equation 2.1, all other options will be masked except E, the only possible output. E token is added to the stack.
- In the second iteration, or any iteration where the token E is popped, the input for all examples and all softmax outputs are masked except the entries representing EET and P according to Rule equation 2.2. If EET is sampled, T, E and E tokens will be added to the stack separately in that exact order to expand the program further. If P is sampled, it will be added to the stack and the program cannot expand further.
- If T is popped out of the stack, the output space for that iteration will be limited to all the operations (Rule equation 2.3). Similarly, if P is popped out, the output space is limited to all the geometric shapes (Rule equation 2.4).
- When a shape token is sampled, it will not be added to the grammar stack as they do not contribute to the program structure. Instead, the image of the shape will be pushed onto the corresponding image stack.
- When an operation token is sampled, it also will not be added to the grammar stack. Instead,

we pop out the top two images to apply the operation on them and push the final image onto the image stack again.

- When the stack has popped out all the added tokens, the end token $ will be popped out in the last iteration. We then finish the sampling as standard RNN language models.

In practice, we implement the masking mechanism by adding a vector to the output before passing it into the logsoftmax layer to get the probability. The vector contains 0 for valid output and large negative numbers for invalid ones. This makes sure that invalid options will have almost zero probability of being sampled. The input of the RNN cell includes encoded target images and intermediate images from the image stack, embedded pop-out tokens from the grammar stack, and the hidden state from the RNN's last iteration. The exact algorithm is in Algorithm 3.

2.3 Experiments

There are two datasets we used for experiments – a synthetic dataset generated by the CFG specified in Section 2.2 and a 2D CAD furniture dataset. We compared the result of our algorithm to that of the same neural network trained with supervision of ground-truth programs. We observed that the supervised model shows poorer generalization in both datasets despite its access to additional ground-truth programs. We provide qualitative and quantitative ablation study of our algorithm on the synthetic dataset. We also showed that our algorithm can *approximate* the CAD images with programs despite not having exact matches. On the CAD furniture dataset, it outperforms the supervised pretrained model and achieves competitive result to the refined model. Additionally, we verify empirically that the stepwise entropy estimator (Equation 2.9) indeed has smaller variance than the naive estimator (Equation 2.8) as proven in Lemma 2.2.1.

Algorithm 3 TreeLSTM Model

Input: Grammar Stack S, Image Stack I, Target Image $\mathbf{x}$, Sample Set $\mathbb{B}$

function TREELSTM($S, I, \mathbb{B}, \mathbf{x}$)
  $\tilde{\mathbf{x}} \leftarrow \text{Encode}(\mathbf{x})$
  for $s^i_{j-1}, \phi_{i,j-1} \in \mathbb{B}$ **do**
    $g_i \leftarrow S_i.\text{pop}()$
    $\tilde{g}_i \leftarrow \text{Embed}(g_i)$
    $\tilde{I}_i \leftarrow \text{Encode}(I_i)$
    $H_{i,j} \leftarrow \text{LSTM}(\tilde{g}_i, \tilde{I}_i, \tilde{\mathbf{x}}, H_{i,j-1})$
    $\mathbf{p_{i,j}} \leftarrow \text{softmax}(f(H_{i,j}) + \text{Mask}(g_i))$
    Estimate entropy at this level:
    $\mathcal{H}_{i,j} = \tilde{\mathbf{p}}_{\mathbf{i,j}} \cdot \log \tilde{\mathbf{p}}_{\mathbf{i,j}}$
    Update the log probabilities of partial sequences
    $\vec{\boldsymbol{\phi}}_{i,j} = \vec{\mathbb{1}} \cdot \phi_{i,j-1} + \log \mathbf{p_{i,j}}$
  end for
  return $\boldsymbol{\Phi}_{:,j}, \mathcal{H}_{i,j}$
end function

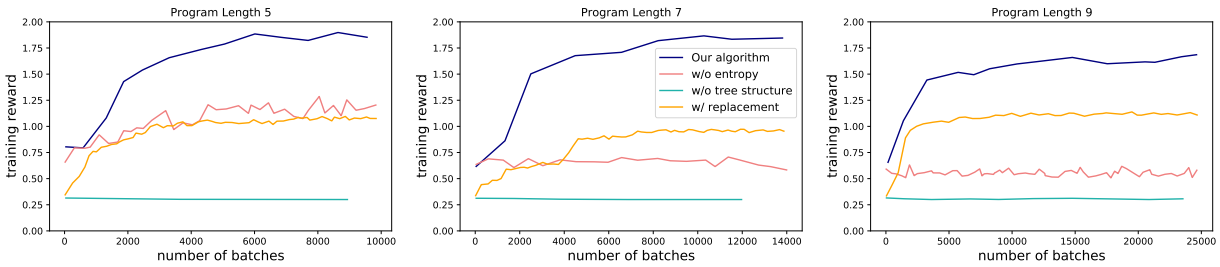


Figure 2.4: From left to right, we have reward per batch for programs of length 5, 7 and 9. It demonstrates the performance of our algorithm and controlled comparison in performance with alternative algorithms by removing one component at a time.

2.3.1 Synthetic Dataset Study

We use three synthetic datasets to test our algorithm. The action space includes 27 shapes (Figure 2.1), 3 operations and 2 grammar non-terminals to create a 64 by 64 images. The search space for an image up to 3 shapes (or program length 5) is around 1.8×10^5 and it gets up to 1.1×10^9 for 5 shapes (or program length 9). We separate our dataset by the length of the program to differentiate images with increasing complexity. Our synthetic dataset is generated by filtering out the duplicates and empty images in combinations of shape and operation actions in text. Images are considered duplicates if only 120 pixels are different between the two and are considered empty if there are no more than 120 pixels in the image. Table 2.1 contains the dataset size information.

Ablation Study of Design Components For these 3 datasets, we sampled 19 programs without replacement for each target image during training. The negative entropy coefficient is 0.05 and the learning rate is 0.01. We use SGD with 0.9 momentum.

Removing either one of the three design components has reduced the performance of our algorithm. Under sampling with replacement setting, the model is quickly stuck at a local optimum (Figure 2.4 (yellow)). Without the entropy term in the objective function, the reward function is only able to improve on the length 5 dataset but fails to do so on longer programs. Both techniques have facilitated exploration that helps the model to escape local minimum. Without tree structure, the reward stays around the lowest reward (Figure 2.4 (green)) because the program is unable to generate valid programs. Grammar encoded tree LSTM effectively constraints the search space such that the sampled programs are valid and can provide meaningful feedback to the model.

We allow variations in the generated programs as long as the target images can be recovered, thus we evaluate the program quality in terms of the reconstructed image's similarity to the target image. We measure our converged algorithm's performance (Table 2.2) on the three test sets with Chamfer and IoU reward metrics (Equation 2.7 first and second term). The perfect match receives 1 in both metrics. Figure 2.5 provides some qualitative examples on the algorithms.

Comparison With Supervised Training We compare a supervised learning method's train and test results on the synthetic dataset, using the same neural network model as in the unsupervised method. The input at each step is the concatenation of embedded ground truth program and the

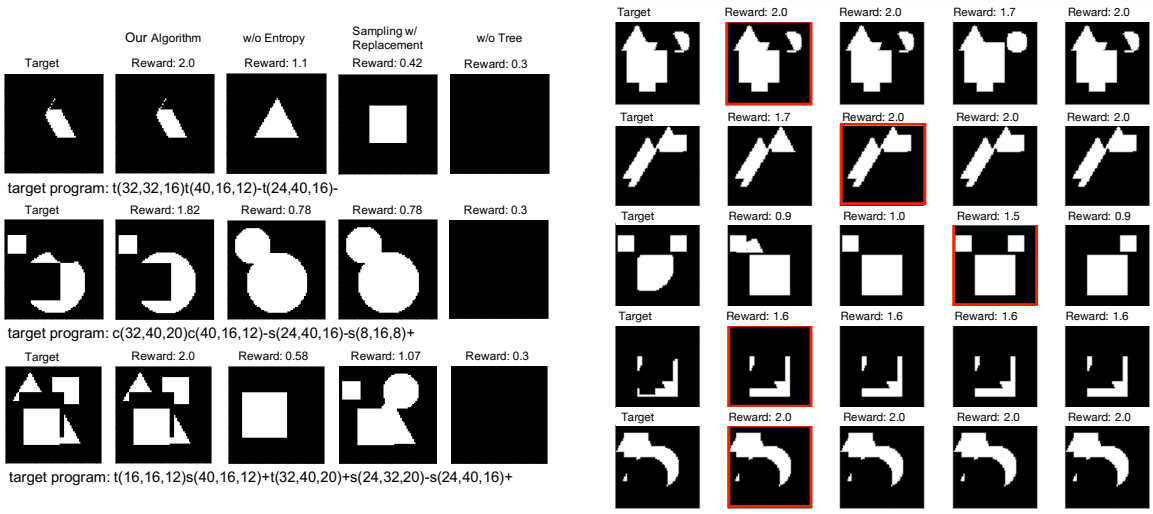


Figure 2.5: a) We show a target image from each dataset and attach its correct program below. To the right are the reconstructed output programs from our algorithm and three variants each removing one design component. The reward is on top of the reconstructed images. (b) Some reconstructed example output programs of our algorithm. Each row represents one data point. The leftmost images of the five columns are the target images and the four columns to their right are the reconstructed outputs of four samples. The final output highlighted in red has the highest reward.

Type	Length 5	Length 7	Length 9	2D CAD
Training set size	3600	4800	12000	10000
Testing set size	586	789	4630	3000

Table 2.1: Dataset statistics.

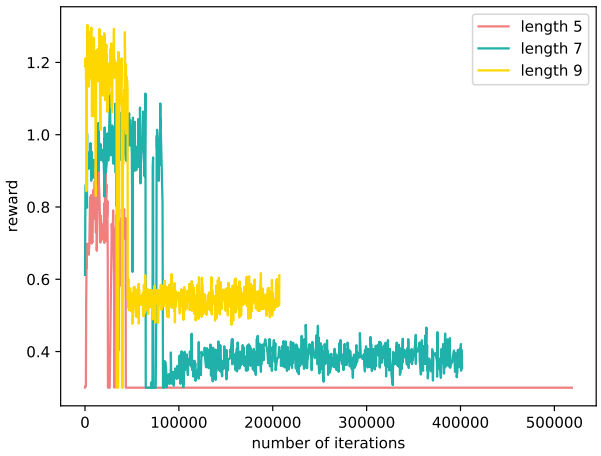


Figure 2.6: REINFORCE fine-tuning with pretrained model.

Test Metric	Length 5	Length 7	Length 9
Chamfer Reward ↑	0.98	0.96	0.96
IoU Reward ↑	0.99	0.96	0.96

Table 2.2: The performance of the converged model of our algorithm on the test set measured with Chamfer reward and IoU reward.

encoded final and intermediate images. We use the same Chamfer reward metric as in Table 2.2 to measure the quality of the programs. The test results of the supervised method worsen with the increasing complexity (program length) while the train results are almost perfect across all three datasets. The unsupervised method receives consistently high scores and generalizes better to new data in comparison to the supervised method (Table 2.3). This phenomenon can be explained by program aliasing [Bunel et al., 2018]. The RL method *treats all correct programs equally* and directly optimizes over the reward function in the image space while the supervised method is limited to the content of the synthetic dataset and only optimizes over the loss function in the program space.

Supervised Pretrained On Limited Data With REINFORCE Fine-tuning In this experiment we pretrained the supervised model on a third of the synthetic training dataset till convergence. We take the model and further fine-tune it with vanilla REINFORCE on the full training sets. We report the reward throughout fine-tuning process (Figure 2.6) and it dropped sharply in all three datasets. Our explanation is that while the output of the original supervised pretrained model follow grammatical structures, they are not able to retain the structure consistently after updates during the refinement process, which leads to the collapse.

2.3.2 2D CAD Furniture Dataset Study

The dataset used in this experiment is a 2D CAD dataset [Sharma et al., 2018] that contains binary 64×64 images of various furniture items. We apply our algorithm to this problem with

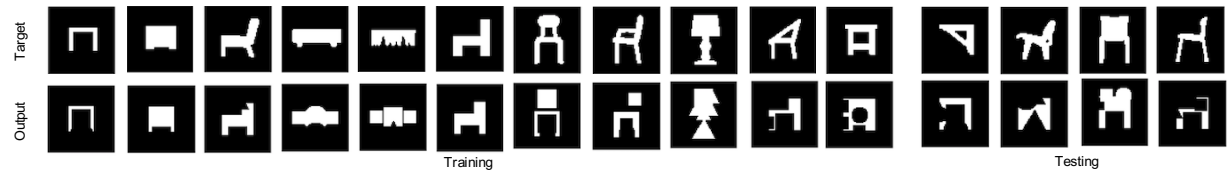


Figure 2.7: The four examples on the right are from the test set, and the rest on the left are from the training set. The target images are on top and the reconstruction from the output programs are at the bottom.

Chamfer Reward ↑	length 5	length 7	length 9
Training	1.00	1.00	0.99
Testing	0.99	0.91	0.83

Table 2.3: Supervised training results.

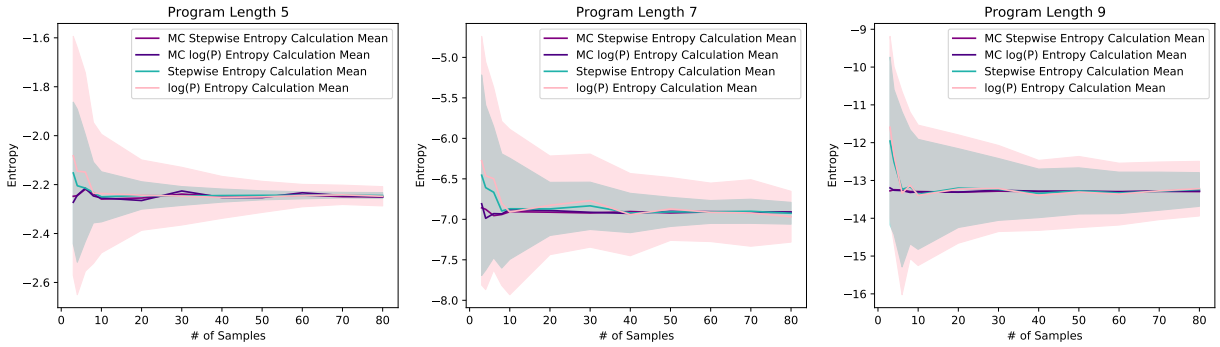


Figure 2.8: Compare the entropy estimation following the Equation **??** as a weighted sum of stepwise entropy versus taking the average of the sequence log probability. The number of samples varies from 2 to 80 on the x-axis. The shaded area represents the standard deviation of each estimator. From left to right, we demonstrate the result on datasets of three program lengths.

an action space of 396 basic shapes plus the operations and grammatical terminals described in Section 2.2. We limit the number of LSTM iterations to 24 steps, which corresponds to a maximum of 6 shapes. For an image up to 6 shapes the search space is 9.4×10^{17}. If we remove the grammar-encoded tree structure, the search space is 3.8×10^{28}. In order to scale up to such a big search space from the synthetic experiment, we increase the number of programs sampled without replacement to 550. The higher number usually corresponds to faster convergence and better performance at convergence but the performance gain diminishes at around 500 for this problem. The learning rate and entropy values used are 0.01 and 0.007 respectively. We train the model with only the Chamfer reward (first part of the Equation 2.7) because these images are not generated by CFG and exact matching solutions do not exist. During training, the reward converges to 0.72. Qualitative results are reported from the train and test set (Figure 2.7). The program reconstructions are able to capture the overall profile of the target images. However, the cutouts and angles deviate from the original because the shape actions consist solely of unrotated squares, perfect circles and equilateral triangles.

Chamfer Distance ↓	k = 1	k =3	k = 5
30k Supervised Model	4.09	3.38	3.02
30k RL Refined Model	1.92	1.83	1.79
30k SWOR RL Refined Model	1.96	1.82	1.77
150k Supervised Model	3.64	2.89	2.63
150k RL Refined Model	1.91	1.79	1.73
150k SWOR RL Refined Model	1.93	1.79	1.73
300k Supervised Model	3.32	2.69	2.38
300k RL Refined Model	1.66	1.54	1.50
300k SWOR RL Refined Model	1.65	1.53	1.49
Unsupervised Model	1.51	1.48	**1.47**

Table 2.4: Empirical comparison of supervised model pre-trained on 30k, 150k and 300k programs, their fine-tuned models and our model on 2D CAD dataset with Chamfer distance

Comparison With Supervised Pretraining In this section, we measure the image similarity directly in Chamfer Distance (CD) (Equation 2.6) for comparison. We pretrained a model on 300k, 150k and 30k ground truth programs (including duplicates) each with learning rate of 0.001. We selected the pretrained models that reaches the lowest CD (at 1.41, 2.00, 2.79 respectively) on a synthetic validation set. We further fine-tuned the pretrained models on the CAD dataset with learning rate 0.006. The transition between pretraining and fine-tuning is delicate. The grammar structure of the output programs (Figure 2.6) collapses when we set the learning rate to be 0.01 (as opposed to 0.006) or fine-tune an unconverged pretrain model. Our model was trained directly on the CAD dataset without supervision with learning rate of 0.01 and entropy coefficient (ec) of 0.009. Entropy coefficient trades off between exploitation and exploration. Higher ec leads to slower convergence but the model is less likely to be stuck at local optimum. Setting ec in the range between 0.005 and 0.012 for this problem has not impact the result significantly. We report the result of the three pretrained models, vanilla RL fine-tuned models, SWOR RL fine-tuned models as well as our model after beam search with $k = 1, 3, 5$ in Table 2.4.

The poor performance of the pre-trained model shows that it is not able to directly generalize to the novel dataset due to the mismatch of the training dataset and the CAD dataset. The unsupervised method exceeds the performance of a supervised model by a large margin because it *was trained directly on the target domain*. It also removes the hyper-parameter tuning step in the transition to the RL fine-tuning and reaches a result competitive to a refined model. The unsupervised method performs better when $k = 1, 3$ than the fine-tuned models. At convergence, the results of the fine-tuned model pre-trained on 300k synthetic data and the unsupervised model become very close at $k = 5$. The two types of refined models reach similar results given the same amount of pretraining synthetic data – poorer performance on less data – confirming that the quality of the pretraining dataset is a limiting factor for the downstream models.

We include additional enlarged test outputs (Figure 2.9). We add the corresponding output program below each target/output pair. We observe that the algorithm approximates thin lines with triangles in some cases. Our hypothesis for the cause is the Chamfer Distance reward function being a greedy algorithm and it finds the matching distance based on the nearest features.

2.3.3 Variance Study of Entropy Estimation

This study (Figure 2.8) investigates the empirical relationship between the two variance estimators and verifies Lemma 2.2.1 that $\hat{\mathcal{H}}_D$ achieves lower variance than $\hat{\mathcal{H}}$ (Section 2.2.2).

We take a single model saved at epoch 40 during the training time of the length 5, 7, and 9 datasets and estimate the entropy with $\hat{\mathcal{H}}_D$ (Equation 2.9) and $\hat{\mathcal{H}}$ (Equation 2.8). We consider two sampling schemes: with and without replacement. We combine both entropy estimation methods with the two sampling schemes creating four instances for comparisons. The x-axis of the plot documents the number of samples to obtain a single estimation of the entropy. We further repeat the estimation 100 times to get the mean and variance. The means of SWR method act as a baseline for the means of the SWOR while we compare the standard deviations (the shaded area) of the two entropy estimation methods.

Across all three datasets, $\hat{\mathcal{H}}_D$ (green) shows significantly smaller variance with the number of samples ranging from 2 to 80. However, we notice that longer programs, or more complex images, require much more samples to reduce variance. This makes sense because the search space increases exponentially with the program length. The initial bias in the SWOR estimation dissipates after the number of samples grows over 10 and is greater in datasets with longer program lengths.

2.4 Discussion

Current program synthesis approaches for non-differentiable renderers employ a two-step scheme that requires the user to first generate a synthetic dataset for pretraining and then use RL fine-tuning with target images. A purely RL-driven approach does not require the curation of a pretraining dataset and can learn directly from target images. Further, unlike approaches that rely on supervised pretraining, the RL approach does not restrict the model to only the program(s) in the synthetic dataset when multiple equal reconstructions exist. This limitation of the pretraining dataset can further impact the downstream models' ability to generalize. In this thesis, we introduced the first unsupervised algorithm capable of parsing CSG images—created by a non-differentiable renderer—into CFG programs without pretraining. We do so by combining three key ingredients to improve the sample efficiency of a REINFORCE-based algorithm: (i) a grammar-encoded tree LSTM to constrain the search space; (ii) entropy regularization for trading off exploration and exploitation; (iii) sampling without replacement from the CFG syntax tree for better estimation. Our ablation study emphasizes the qualitative and quantitative contributions of each design component. Even though our RL approach does not have access to a pretraining dataset it achieves stronger performance than a supervised method on the synthetic 2D CSG dataset. It further outperforms the supervised pre-trained model on the novel 2D CAD dataset and is competitive with the RL fine-tuned model.

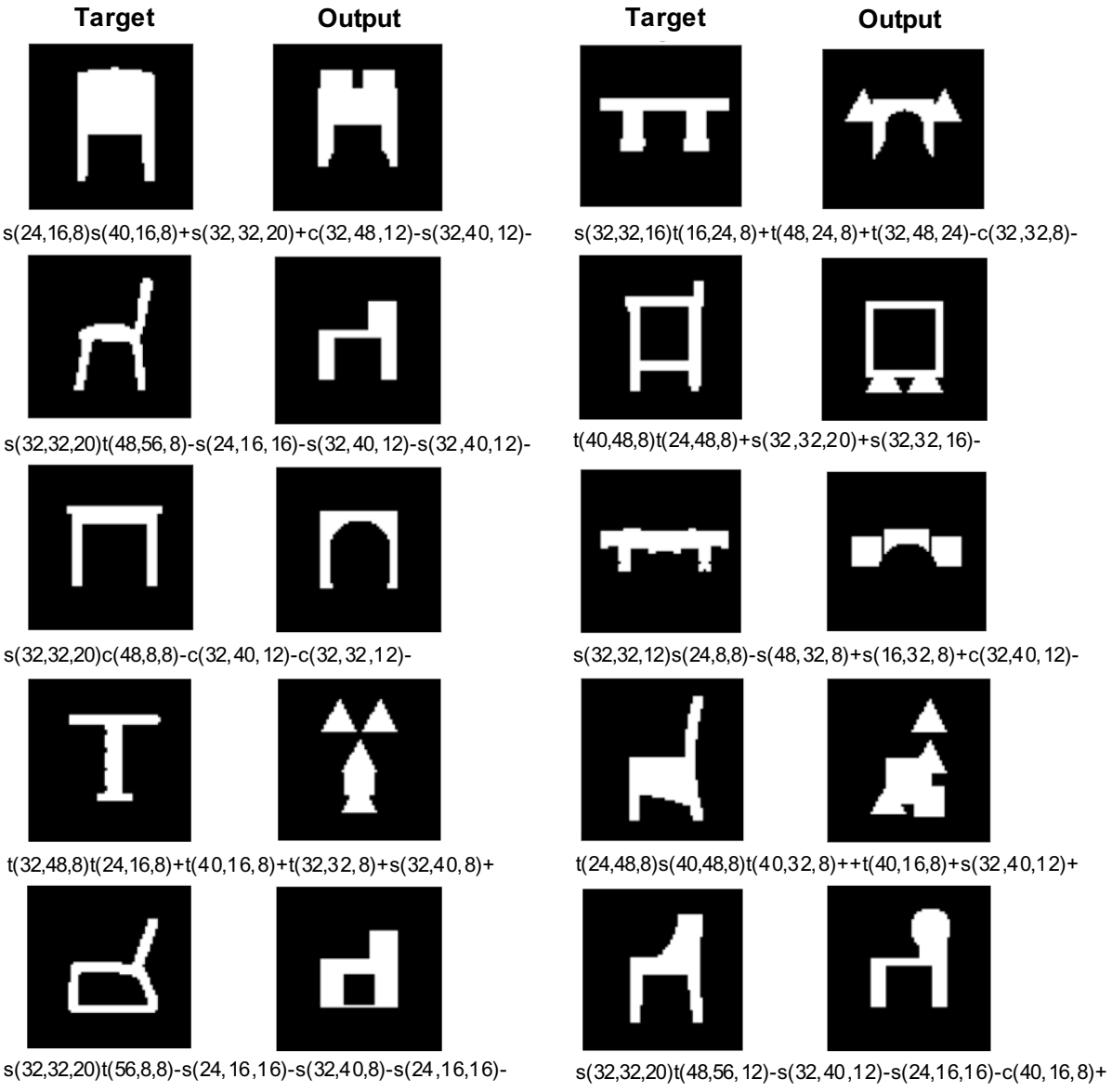


Figure 2.9: Additional test output with corresponding programs. The odd-numbered columns contain the target images and the images to their right are example outputs.

Chapter 3

Variational Inference on Low-Dimensional Data

3.1 Introduction

Variational autoencoders (VAEs) have recently enjoyed a revived interest, both due to architectural choices that have led to improvements in sample quality [Oord et al., 2017, Razavi et al., 2019b, Vahdat and Kautz, 2020] and due to algorithmic insights [Dai et al., 2017, Dai and Wipf, 2019]. Nevertheless, fine-grained understanding of the behavior of VAEs is lacking, both on the theoretical and empirical level.

In our paper, we study a common setting of interest where the data is supported on a low-dimensional manifold — often argued to be the setting relevant to real-world image and text data due to the *manifold hypothesis* (see e.g. Goodfellow et al. [2016]). In this setting, Dai and Wipf [2019] proposed a two-stage training process for VAEs, based on a combination of empirical and theoretical arguments suggesting that for standard VAE training with such data distributions: (1) the generator's covariance will converge to 0, (2) the generator will learn the correct manifold, but not the correct density on the manifold (3) the number of approximately 0 eigenvalues in the encoder covariance will equal the intrinsic dimensionality of the manifold (see also Dai et al. [2017]).

In this paper, we revisit this setting and explore the behaviour of both the VAE loss, and the training dynamics. Through a combination of theory and experiments we show that:

- In the case of the data manifold being **linear** (i.e. the data is Gaussian, supported on a linear subspace—equivalently, it is produced as the pushforward of a Gaussian through a linear map), and the encoder/decoder being parametrized as linear maps, we show that: a) the set of optima includes parameters for which the generator's support is a *strict superset* of the data manifold; b) the gradient descent dynamics are such that they converge to generators with support *equal* to the support of the data manifold. This provides a full proof of the conjecture in Dai and Wipf [2019], albeit we show the phenomenon is a combination of both the *location* of the minima of the loss as well as an *implicit bias of the training dynamics*.
- In the case of the data manifold being **nonlinear** (i.e. the data distribution is the pushforward of the Gaussian through a nonlinear map $f : \mathbb{R}^r \to \mathbb{R}^d, r \leq d$), the gradient descent

dynamics from a random start often converges to generators G whose support *strictly contains* the support of the underlying data distribution, while driving reconstruction error to 0 and driving the VAE loss to $-\infty$. This shows that the conjecture in Dai and Wipf [2019] does not hold for general nonlinear data manifolds and architectures for the generator/encoder.

3.1.1 Related work

Implicit regularization. Interestingly, the implicit bias towards low-rank solutions in the VAE which we discover is consistent with theoretical and experimental results in other settings, such as deep linear networks/matrix factorization (e.g. Gunasekar et al. [2018], Li et al. [2018b], Arora et al. [2019], Li et al. [2020], Jacot et al. [2021]), although it seems to be for a different mathematical reason — unlike those settings, initialization scale does not play a major role. Similar to the setting of implicit margin maximization (see e.g. Ji and Telgarsky [2018], Schapire and Freund [2013], Soudry et al. [2018]), in our VAE setting the optima are asymptotic (though approaching a finite point, not off at infinity) and the loss goes to $-\infty$. Kumar and Poole [2020], Tang and Yang [2021] also explore some implicit regularization effects tied to the Jacobian of the generator and the covariance of the Gaussian noise.

Architectural and Algorithmic Advances for VAEs. There has been a recent surge in activity with the goal of understanding VAE training and improving its performance in practice. Much of the work has been motivated by improving posterior modeling to avoid problems such as "posterior collapse", see e.g. [Dai et al., 2020, Razavi et al., 2019a, Pervez and Gavves, 2021, Lucas et al., 2019, He et al., 2019, Oord et al., 2017, Razavi et al., 2019b, Vahdat and Kautz, 2020]. Most relevant to the current work are probably the works Dai and Wipf [2019] and Lucas et al. [2019] discussed earlier. A relevant previous work to these is Dai et al. [2017]; one connection to the current work is that they also performed experiments with a ground truth manifold, in their case given as the pushforward of a Gaussian through a ReLU network. In their case, they found that for a certain decoder and encoder architectures that they could recover the intrinsic dimension using a heuristic related to the encoder covariance eigenvalues from Dai and Wipf [2019]; our results are complementary in that they show that this phenomena is not universal and does not hold for other natural datasets (e.g. manifold data on a sphere fit with a standard VAE architecture).

3.2 Setup

We will study the behavior of VAE learning when data lies on a low-dimensional manifold—more precisely, we study when the generator can recover the support of the underlying data distribution. In order to have a well-defined "ground truth", both for our theoretical and empirical results, we will consider synthetic dataset that are generated by a "ground truth" generator as follows.

Data distribution: To generate a sample point x for the data distribution, we will sample $z \sim N(0, I_{r^*})$, and output $x = f(z)$, for a suitably chosen f. In the *linear* case, $f(z) = Az$, for

some matrix $A \in \mathbb{R}^{d \times r^*}$. In the *nonlinear* case, $f(z)$ will be a nonlinear function $f : \mathbb{R}^{r^*} \to \mathbb{R}^d$. We will consider several choices for f.

Parameterization of the trained model: For the model we are training, the generator will sample $z \sim N(0, I_r)$ and output $x \sim N(f(z), \epsilon I)$, for trainable f, ϵ; the encoder given input x will output $z \sim N(g(x), D)$, where $D \in \mathbb{R}^{r \times r}$ is a diagonal matrix, and g, D are trainable. In the linear case, f, g will be parameterized as matrices $\tilde{A}, \tilde{B}$; in the nonlinear case, several different parameterizations will be considered. In either case the VAE Loss will be denoted by $L(\cdot)$, see equation 3.3.

3.3 Our Results

Linear VAEs: the correct distribution is not recovered. Recall in the linear case, we train a linear encoder and decoder to learn a Gaussian distribution consisting of data points $x \sim N(0, \Sigma)$ — equivalently, the data distribution is the pushforward of a standard Gaussian $z \sim N(0, I_{r^*})$ through a linear generator $x = Az$ with $AA^T = \Sigma$; see also Section 3.2 above. In Theorem 1 of Lucas et al. [2019], the authors proved that in a certain probabilistic PCA setting where Σ is full-rank, the landscape of the VAE loss has no spurious local minima: at any global minima of the loss, the VAE decoder exactly matches the ground truth distribution, i.e. $\tilde{A}\tilde{A}^T + \epsilon^2 I = \Sigma$.

We revisit this problem in the setting where Σ has rank less than d so that the data lies on the *lower-dimensional manifold/subspace* spanned by the columns of A or equivalently Σ, denoted span(A). We show empirically (i.e. via simulations) in Section 3.6 that when Σ is rank-degenerate the VAE actually *fails to recover the correct distribution*. More precisely, the recovered $\tilde{A}$ has the correct column span but fails to recover the correct density — confirming predictions made in Dai and Wipf [2019]. We then explain theoretically why this happens, where it turns out we find some surprises.

Landscape Analysis: Linear and Nonlinear VAE. Dai and Wipf [2019] made their predictions on the basis of the following observation about the *loss landscape*: there can exist sequences of VAE solutions whose objective value approaches $-\infty$ (i.e. are asymptotic global minima), for which the generator has the correct column span, but does not recover the correct density on the subspace. They also informally argued that these are all of the asymptotic global minima of loss landscape (Pg 7 and Appendix I in Dai and Wipf [2019]), but did not give a formal theorem or proof of this claim.

We settle the question by showing this is *not the case*:[1] namely, there exist many convergent sequences of VAE solutions which still go to objective value $-\infty$ but also do not recover the correct column span — instead, the span of such $\tilde{A}$ is a strictly larger subspace. More precisely, we obtain a tight characterization of all asymptotic global minima of the loss landscape:

Theorem 3.3.1 (Informal Theorem on Optima of Linear VAE Loss)**.** *Suppose that $\tilde{A}, \tilde{B}$ are fixed matrices such that $A = \tilde{A}\tilde{B}A$ and suppose that $\#\{i : \tilde{A}_i = 0\} > r - d$, i.e. the number of zero*

[1]They also argued this would hold in the nonlinear case, but our simulations show this is generally false in that setting, even for the solutions chosen by gradient descent with a standard initialization — see Section 3.6.

columns of $\tilde{A}$ is strictly larger than $r-d$. Then there exists $\tilde{\epsilon}_t \to 0$ and positive diagonal matrices $\tilde{D}_t$ such that $\lim_{t\to\infty} L(\tilde{A}, \tilde{B}, \tilde{D}_t, \tilde{\epsilon}_t) = -\infty$. Also, these are all of the asymptotic global minima: any convergent sequence of points $(\tilde{A}_t, \tilde{B}_t, \tilde{D}_t, \tilde{\epsilon}_t)$ along which the loss L goes to $-\infty$ satisfies $\tilde{A}_t \to \tilde{A}, \tilde{B}_t \to \tilde{B}$ with $A = \tilde{A}\tilde{B}A$ such that $\#\{i : \tilde{A}_i = 0\} > r - d$.

To interpret the constraint $\#\{i : \tilde{A}_i = 0\} > r - d$, observe that if the data lies on a lower-dimensional subspace of dimension $r^* < d$ (i.e. r^* is the rank of Σ), then there exists a generator which generates the distribution with $r - r^* > r - d$ zero columns by taking an arbitrary low-rank factorization $LL^T = \Sigma$ with $L : d \times r^*$ and defining $A : d \times r$ by $A = \begin{bmatrix} L & 0_{d\times r-r^*} \end{bmatrix}$. The larger the gap is between the manifold/intrinsic dimension r^* and the ambient dimension d, the more flexibility we have in constructing asymptotic global minima of the landscape. Also, we note there is no constraint in the Theorem that $r - d \geq 0$: the assumption is automatically satisfied if $r < d$.

To summarize, the asymptotic global minima satisfy $A = \tilde{A}\tilde{B}A$, so the column span of $\tilde{A}$ contains that of A, but in general it can be a higher dimensional space. For example, if $d, r \geq r^*+2$ and and the ground truth generator is $A = \begin{bmatrix} I_{r^*} & 0 \\ 0 & 0 \end{bmatrix}$, then $\tilde{A} = \begin{bmatrix} I_{r^*+1} & 0 \\ 0 & 0 \end{bmatrix}$ and $\tilde{B} = \begin{bmatrix} I_{r^*+1} & 0 \\ 0 & 0 \end{bmatrix}$ is a perfectly valid asymptotic global optima of the landscape, even though decoder $\tilde{A}$ generates a different higher-dimensional Gaussian distribution $N\left(0, \begin{bmatrix} I_{r^*+1} & 0 \\ 0 & 0 \end{bmatrix}\right)$ than the ground truth. In the above result we showed that there are asymptotic global minima with higher dimensional spans even with the common restriction that the encoder variance is diagonal; if we considered the case where the encoder variance is unconstrained, as done in Dai and Wipf [2019], and/or can depend on its input x, this can only increase the number of ways to drive the objective value to $-\infty$.

We also consider the analogous question in the *nonlinear* VAE setting where the data lies on a low-dimensional manifold. We prove in Theorem 3.4.6 that even in a very simple example where we fit a VAE to generate data produced by a 1-layer ground truth generator, there exists a bad solution with strictly larger manifold dimension which drives the reconstruction error to zero (and VAE loss to $-\infty$). The proof of this result does not depend strongly on the details of this setup and it can be adapted to construct bad solutions for other nonlinear VAE settings.

We note that the nature both of these result is asymptotic: that is, they consider sequences of solutions whose loss converges to $-\infty$ — but not the rate at which they do so. In the next section, we will consider the trajectories the optimization algorithm takes, when the loss is minimized through gradient descent.

Linear VAE: implicit regularization of gradient flow. The above theorem indicates that studying the minima of the loss landscape alone cannot explain the empirical phenomenon of linear VAE training recovering the support of the ground truth manifold in experiments; the only prediction that can be made is that the VAE will recover a *possibly larger* manifold containing the data.

We resolve this tension by proving that *gradient flow*, the continuous time version of gradient descent, has an *implicit bias* towards the low-rank global optima. Precisely, we measure the effective rank quantitatively in terms of the singular values: namely, if $\sigma_k(\tilde{A})$ denotes the k-th largest singular value of matrix $\tilde{A}$, we show that all but the largest $\dim(\operatorname{span} A)$ singular values

of $\tilde{A}$ decay at an exponential rate, as long as: (1) the gradient flow continues to exist[2] , and (2) the gradient flow does not go off to infinity, i.e. neither $\tilde{A}$ or $\tilde{\epsilon}$ go to infinity (in simulations, the decoder $\tilde{A}$ converges to a bounded point and $\tilde{\epsilon} \to 0$ so the latter assumption is true). To simplify the proof, we work with a slightly modified loss which "eliminates" the encoder variance by setting it to its optimal value: $L_1(\tilde{A}, \tilde{B}, \tilde{\epsilon}) := \min_{\tilde{D}} L(\tilde{A}, \tilde{B}, \tilde{\epsilon}, \tilde{D})$; this loss has a simpler closed form, and we believe the theorems should hold for the standard loss as well. (Generally, gradient descent on the original loss L will try to optimize $\tilde{D}$ in terms of the other parameters, and if it succeeds to keep $\tilde{D}$ well-tuned in terms of $\tilde{A}, \tilde{B}, \tilde{\epsilon}$ then L will behave like the simplified loss L_1.)

Theorem 3.3.2 (Informal Theorem on the Implicit Bias of Gradient Flow)**.** *Let $A : d \times r$ be arbitrary and define W to be the span of the rows of A, let $\tilde{\Theta}(0) = (\tilde{A}(0), \tilde{B}(0), \tilde{\epsilon}(0))$ be an arbitrary initialization and define the gradient flow $\tilde{\Theta}(t) = (\tilde{A}(t), \tilde{B}(t), \tilde{\epsilon}(t))$ by the ordinary differential equation (ODE)*

$$\frac{d\tilde{\Theta}(t)}{dt} = -\nabla L_1(\tilde{\Theta}(t)) \tag{3.1}$$

with initial condition Θ_0. If the solution to this equation exists on the time interval $[0, T]$ and satisfies $\max_{t\in[0,T]} \max_j [\|(\tilde{A}_t)_j\|^2 + \tilde{\epsilon}_t^2] \leq K$, then for all $t \in [0, T]$ we have

$$\sum_{k=\dim(W)+1}^{d} \sigma_k^2(\tilde{A}(t)) \leq C(A, \tilde{A})\, e^{-t/K} \tag{3.2}$$

where $C(A, \tilde{A}) := \|P_{W^\perp} \tilde{A}^T(0)\|_F^2$ and $P_{W^\perp}$ is the orthogonal projection onto the orthogonal complement of W.

Together, our Theorem 3.3.1 and Theorem 3.3.2 show that if gradient descent converges to a point while driving the loss to $-\infty$, then it successfully recovers the ground truth subspace/manifold $\operatorname{span} A$. This shows that, in the linear case, the conjecture of Dai and Wipf [2019] can indeed be validated provided we incorporate training dynamics into the picture. The prediction of theirs we do not prove is that the number of zero entries of the encoder covariance D converges to the intrinsic dimension; as shown in Table 3.1, in a few experimental runs this does not occur — in contrast, Theorem 3.3.2 implies that $\tilde{A}$ should have the right number of nonzero singular values and our experiments agree with this.

Nonlinear VAE: Dynamics and Simulations. In the linear case, we showed that the implicit bias of gradient descent leads the VAE training to converge to a distribution with the correct support. In the nonlinear case, we show that this does not happen—even in simple cases.

Precisely, in the setup of the one-layer ground truth generator, where we proved (Theorem 3.4.6) there exist bad global optima of the landscape, we verify experimentally (see Figure 3.1) that gradient descent from a random start does indeed converge to such bad asymptotic minima. In particular, this shows that whether or not gradient descent has a favorable implicit bias strongly depends on the data distribution and VAE architecture.

[2]We remind the reader that the gradient flow on loss $L(x)$ is a differential equation $dx/dt = -\nabla L(x)$. Unlike discrete-time gradient descent, gradient flow in some cases (e.g. $dx/dt = x^2$) has solutions which exist only for a finite time (e.g. $x = 1/(1-t)$), which "blows up" at $t = 1$), so we need to explicitly assume the solution exists up to time T.

More generally, by performing experiments with synthetic data of known manifold dimension, we make the following conclusions: (1) gradient descent training recovers manifolds approximately containing the data, (2) these manifolds are generally not the same dimension as the ground truth manifold, but larger (this is in contrast to the conjecture in Dai and Wipf [2019] that they should be equal) even when the decoder and encoder are large enough to represent the ground truth and the reconstruction error is driven to 0 (VAE loss is driven to $-\infty$), and (3) of all manifolds containing the data, gradient descent still seems to favor those with relatively low (but not always minimal) dimension. Further investigating the precise role of VAE architecture and optimization algorithm, as well as the interplay with the data distribution is an exciting direction for future work.

3.4 VAE Landscape Analysis

In this section, we analyze the landscape of a VAE, both in the linear and non-linear case.

Preliminaries and notation. We use a VAE to model a datapoint $x \in \mathbb{R}^d$ as the pushforward of $z \sim N(0, I_r)$. We have the following standard VAE architecture:

$$p(x|z) = N(f(z), \epsilon^2 I), \qquad q(z|x) = N(g(x), D)$$

where $\epsilon^2 > 0$ is the decoder variance, D is a diagonal matrix with nonnegative entries, and f, g, D, ϵ are all trainable parameters. (For simplicity, our D does not depend on x; this is the most common setup in the linear VAE case we will primarily focus on.) The VAE objective (see Lemma **??** for explicit derivation) is to *minimize*:

$$\begin{aligned} L(f, g, D, \epsilon) := \mathbb{E}_{x\sim p^*}\mathbb{E}_{z'\sim N(0,I_r)}\Big[\frac{1}{2\epsilon^2}\|x - f(g(x) + D^{1/2}z')\|^2 + \|g(x)\|^2/2\Big] \\ + d\log(\epsilon) + \mathrm{Tr}(D)/2 - \frac{1}{2}\sum_i \log D_{ii}. \end{aligned} \tag{3.3}$$

We also state a general fact about VAEs for the case that the objective value can be driven to $-\infty$, which was observed in [Dai and Wipf, 2019]: they must satisfy $\epsilon \to 0$ and achieve perfect limiting reconstruction error. The first claim in this Lemma is established in the proof of Theorem 4 and the second claim is Theorem 5 in Dai and Wipf [2019].

Lemma 3.4.1 (Theorems 4 and 5 of Dai and Wipf [2019])**.** *Suppose $f_t, g_t, D_t, \epsilon_t$ for $t \geq 1$ are a sequence such that $\lim_{t\to\infty} L(f_t, g_t, D_t, \epsilon_t) = -\infty$. Then: 1) $\lim_{t\to\infty} \epsilon_t = 0$ and 2) $\lim_{t\to\infty} \mathbb{E}_{x\sim p^*}\mathbb{E}_{z'\sim N(0,I_r)}\|x - f_t(g_t(x) + D_t^{1/2}z')\|^2 = 0$.*

In fact, the reconstruction error and ϵ are closely linked in a simple way:

Lemma 3.4.2. *If f, g, D are fixed, then the optimal value of ϵ to minimize L is given by*

$$\epsilon = \sqrt{\frac{1}{d}\mathbb{E}_{x\sim p^*}\mathbb{E}_{z'\sim N(0,I_r)}\left[\|x - f(g(x) + D^{1/2}z')\|^2\right]}.$$

Proof of Lemma 3.4.1. For completeness, we include the proof of these claims; they are similar to the proofs of Theorems 4 and 5 in Dai and Wipf [2019].

First, consider the objective for fixed f, g, D, ϵ and omit the subscript t. We have

$$\mathbb{E}_{x\sim p^*}\mathbb{E}_{z'\sim N(0,I_r)}\Big[\frac{1}{2\epsilon^2}\|x - f(g(x) + D^{1/2}z')\|^2 + \|g(x)\|^2/2\Big] \geq 0$$

and

$$\mathrm{Tr}(D)/2 - \frac{1}{2}\sum_i \log D_{ii} = \frac{1}{2}\sum_i (D_{ii} - \log D_{ii}) \geq r/2$$

from the inequality $x - \log x \geq 1$ for $x \geq 0$. Since these terms are both bounded above, the only way the objective goes to negative infinity is if $d\log\epsilon \to -\infty$ which means $\epsilon \to 0$.

Now that we know $\epsilon_t \to 0$, we claim that $\lim_{t\to\infty} \mathbb{E}_{x\sim p^*}\mathbb{E}_{z'\sim N(0,I_r)}\|x - f_t(g_t(x) + D_t^{1/2}z')\|^2 = 0$. Suppose otherwise: then this for infinitely many t this quantity is lower bounded by some constant $c > 0$, hence the objective for those t is lower bounded by $c/\epsilon^2 + d\log(\epsilon) + r/2$ and this goes to $+\infty$ as $\epsilon \to 0$, instead of $-\infty$. □

Proof of Lemma 3.4.2. Taking the partial derivative of equation 3.3 with respect to ϵ and setting it to zero gives

$$0 = -\frac{1}{\epsilon^3}\mathbb{E}_{x\sim p^*}\mathbb{E}_{z'\sim N(0,I_r)}\|x - f(g(x) + D^{1/2}z')\|^2 + \frac{d}{\epsilon}$$

and solving for ϵ gives the result. □

3.4.1 Linear VAE

Setup: In the linear VAE case, we assume the data is generated from the model $x = Az$ with $A \in \mathbb{R}^{d\times r^*}$ and $z \sim \mathcal{N}(0, I_{r^*})$. We will denote the training parameters by $\tilde{A} \in \mathbb{R}^{d\times r}$, $\tilde{B} \in \mathbb{R}^{r\times d}$, $\tilde{D} \in \mathbb{R}^{r\times r}$, and $\tilde{\epsilon} > 0$, where $r \geq 1$ is a fixed hyperparameter which corresponds to the latent dimension in the trained generator, and we assume $\tilde{D}$ is a diagonal matrix. With this notation in place, the implied VAE has generator/decoder $\tilde{x} \sim \mathcal{N}(\tilde{A}z, \tilde{\epsilon}^2 I_d)$ and encoder $\tilde{z} \sim \mathcal{N}(\tilde{B}x, \tilde{D})$. By Lemma 3.4.3, the VAE objective as a function of parameters $\tilde{\Theta} = (\tilde{A}, \tilde{B}, \tilde{D}, \tilde{\epsilon})$ is:

$$L(\tilde{\Theta}) = \frac{1}{2\tilde{\epsilon}^2}\|A - \tilde{A}\tilde{B}A\|_F^2 + \frac{1}{2}\|\tilde{B}A\|_F^2 + d\log\tilde{\epsilon} + \frac{1}{2}\sum_i \Big(\tilde{D}_{ii}\|\tilde{A}_i\|^2/\tilde{\epsilon}^2 + \tilde{D}_{ii} - \log\tilde{D}_{ii}\Big) \quad (3.4)$$

Lemma 3.4.3. *For the linear VAE as described in Section 3.4.1, the VAE loss can be written as*

$$L(\tilde{\Theta}) = \frac{1}{2\tilde{\epsilon}^2}\|A - \tilde{A}\tilde{B}A\|_F^2 + \frac{1}{2}\|\tilde{B}A\|_F^2 + d\log\tilde{\epsilon} + \frac{1}{2}\sum_i \Big(\tilde{D}_{ii}\|\tilde{A}_i\|^2/\tilde{\epsilon}^2 + \tilde{D}_{ii} - \log\tilde{D}_{ii}\Big) \quad (3.5)$$

Proof. Plugging in the linear VAE parameters into the loss function, we get

$$L(\tilde{A}, \tilde{B}, \tilde{D}, \tilde{\epsilon}) := \mathbb{E}_{x\sim p^*}\mathbb{E}_{z'\sim N(0,I_{\tilde{r}})}\Big[\frac{1}{2\tilde{\epsilon}^2}\|x - \tilde{A}(\tilde{B}x + \tilde{D}^{1/2}z')\|^2 + \|\tilde{B}x\|^2/2\Big] \quad (3.6)$$

$$+ d\log(\tilde{\epsilon}) + \mathrm{Tr}(\tilde{D})/2 - \frac{1}{2}\sum_i \log \tilde{D}_{ii} \tag{3.7}$$

We can write out the expectation as:

$$\begin{aligned}
&\mathbb{E}_{z\sim N(0,I)}\mathbb{E}_{z'\sim N(0,I_{\tilde{r}})}\Big[\frac{1}{2\tilde{\epsilon}^2}\|Az - \tilde{A}(\tilde{B}Az + \tilde{D}^{1/2}z')\|^2 + \|\tilde{B}Az\|^2/2\Big] \\
&= \mathbb{E}_{z\sim N(0,I)}\mathbb{E}_{z'\sim N(0,I_{\tilde{r}})}\Big[\frac{1}{2\tilde{\epsilon}^2}\|(A - \tilde{A}\tilde{B}A)z - \tilde{A}\tilde{D}^{1/2}z'\|^2 + \|\tilde{B}Az\|^2/2\Big] \\
&= \frac{1}{2\tilde{\epsilon}^2}\|A - \tilde{A}\tilde{B}A\|_F^2 + \frac{1}{2\tilde{\epsilon}^2}\|\tilde{A}\tilde{D}^{1/2}\|_F^2 + \frac{1}{2}\|\tilde{B}A\|_F^2
\end{aligned}$$

where we used that z, z' are independent and the identity $\mathbb{E}_{z\sim N(0,I)}\|Mz\|^2 = \langle MM^T, I\rangle = \|M\|_F^2$. Next, we can observe that

$$\|\tilde{A}\tilde{D}^{1/2}\|_F^2 = \sum_i \tilde{D}_{ii}\|\tilde{A}_i\|^2$$

where $\tilde{A}_i$ is the i-th column of the matrix $\tilde{A}$. Therefore we recover equation 3.5. □

Our analysis makes use of a simplified objective L_1, which "eliminates" D out of the objective by plugging in the optimal value of D for a choice of the other variables. We use this as a technical tool when analyzing the landscape of the original loss L.

Lemma 3.4.4 (Deriving the simplified loss L_1)**.** *Suppose that $\tilde{A}, \tilde{B}, \tilde{\epsilon}$ are fixed. Then the objective L is minimized by choosing for all i that $\tilde{D}_{ii} = \frac{\tilde{\epsilon}^2}{\|\tilde{A}_i\|^2+\tilde{\epsilon}^2}$ where $\tilde{A}_i$ is column i of $\tilde{A}$, and for $L_1(\tilde{A}, \tilde{B}, \tilde{\epsilon}) := \min_{\tilde{D}} L(\tilde{A}, \tilde{B}, \tilde{D}, \tilde{\epsilon})$ it holds that*

$$L_1(\tilde{A}, \tilde{B}, \tilde{\epsilon}) = \frac{1}{2\tilde{\epsilon}^2}\|A - \tilde{A}\tilde{B}A\|_F^2 + \frac{1}{2}\|\tilde{B}A\|_F^2 + (d-r)\log\tilde{\epsilon} + \sum_i \frac{1 + \log\left(\|\tilde{A}_i\|^2 + \tilde{\epsilon}^2\right)}{2}. \tag{3.8}$$

Proof. Taking the partial derivative with respect to $\tilde{D}_{ii}$ gives $0 = \|\tilde{A}_i\|^2/\tilde{\epsilon}^2 + 1 - 1/\tilde{D}_{ii}$ which means

$$\tilde{D}_{ii} = \frac{1}{\|\tilde{A}_i\|^2/\tilde{\epsilon}^2 + 1} = \frac{\tilde{\epsilon}^2}{\|\tilde{A}_i\|^2 + \tilde{\epsilon}^2}$$

hence

$$\tilde{D}_{ii}\|\tilde{A}_i\|^2/\tilde{\epsilon}^2 + \tilde{D}_{ii} - \log\tilde{D}_{ii} = 1 - \log\frac{\tilde{\epsilon}^2}{\|\tilde{A}_i\|^2 + \tilde{\epsilon}^2}.$$

It follows that the objective value at the optimal D is

$$\begin{aligned}
L_1(\tilde{A}, \tilde{B}, \tilde{\epsilon}) &= \frac{1}{2\tilde{\epsilon}^2}\|A - \tilde{A}\tilde{B}A\|_F^2 + \frac{1}{2}\|\tilde{B}A\|_F^2 + d\log\tilde{\epsilon} + \frac{1}{2}\sum_i\left(1 - \log\frac{\tilde{\epsilon}^2}{\|\tilde{A}_i\|^2 + \tilde{\epsilon}^2}\right) \\
&= \frac{1}{2\tilde{\epsilon}^2}\|A - \tilde{A}\tilde{B}A\|_F^2 + \frac{1}{2}\|\tilde{B}A\|_F^2 + (d-r)\log\tilde{\epsilon} + \frac{1}{2}\sum_i\left(1 - \log\frac{1}{\|\tilde{A}_i\|^2 + \tilde{\epsilon}^2}\right).
\end{aligned}$$

□

Taking advantage of this simplified formula, we can then identify (for the original loss L) simple sufficient conditions on $\tilde{A}, \tilde{B}$ which ensure they can be used to approach the population loss minimum by picking suitable $\tilde{\epsilon}_t$, $\tilde{D}_t$ and prove matching necessary conditions.

Theorem 3.4.5. *First, suppose that $\tilde{A} : d \times r, \tilde{B} : r \times d$ are fixed matrices such that $A = \tilde{A}\tilde{B}A$ and suppose that $\#\{i : \tilde{A}_i = 0\} > r - d$, i.e. the number of zero columns of $\tilde{A}$ is strictly larger than $r - d$. Then for any sequence of positive $\tilde{\epsilon}_t \to 0$ there exist a sequence of positive diagonal matrices $\tilde{D}_t$ such that:*

1. *For every i such that $\tilde{A}_i \neq 0$, i.e. column i of $\tilde{A}$ is nonzero, we have $(\tilde{D}_t)_{ii} \to 0$.*
2. *$\lim_{t\to\infty} L(\tilde{A}, \tilde{B}, \tilde{D}_t, \tilde{\epsilon}_t) = -\infty$.*

Conversely, suppose that that $\tilde{A}_t, \tilde{B}_t, \tilde{D}_t, \tilde{\epsilon}_t$ is an arbitrary sequence such that $\lim_{t\to\infty} L(\tilde{A}_t, \tilde{B}_t, \tilde{D}_t, \tilde{\epsilon}_t) = -\infty$. Then as $t \to \infty$, we must have that:

1. *$\tilde{\epsilon}_t \to 0$ and $\|A - \tilde{A}_t\tilde{B}_tA\|_F^2 \to 0$.*
2. *$\max_i(\tilde{D}_t)_{ii}\|(\tilde{A}_t)_i\|_F^2 \to 0$ where $(\tilde{A}_t)_i$ denotes the i-th column of $\tilde{A}_t$.*
3. *For any $\delta > 0$, $\liminf_{t\to\infty} \#\{i : \|(\tilde{A}_t)_i\|_2^2 < \delta\} > r - d$, i.e. asymptotically $\tilde{A}_t$ has strictly more than $r - d$ columns arbitrarily close to zero.*

In particular, if $(\tilde{A}_t, \tilde{B}_t, \tilde{D}_t, \tilde{\epsilon}_t)$ converge to a point $(\tilde{A}, \tilde{B}, \tilde{D}, \tilde{\epsilon})$ then $\tilde{\epsilon} = 0$, $A = \tilde{A}\tilde{B}A$, $\tilde{D}_{ii} = 0$ for every i such that $\tilde{A}_i \neq 0$, and $\#\{i : \tilde{A}_i = 0\} > r - d$.

Proof of Theorem 3.4.5. First we prove the sufficiency direction, i.e. that if $A = \tilde{A}\tilde{B}A$ and there exists i such that $\tilde{A}_i = 0$ then we show how to drive the loss to $-\infty$. By Lemma 3.4.4, if we make the optimal choice of D (which clearly satisfies the conditions on D described in the Lemma) the objective simplifies to

$$\begin{aligned} L_1(\tilde{A}, \tilde{B}, \tilde{\epsilon}) &= \frac{1}{2\tilde{\epsilon}^2}\|A - \tilde{A}\tilde{B}A\|_F^2 + \frac{1}{2}\|\tilde{B}A\|_F^2 + (d-r)\log\tilde{\epsilon} + \frac{1}{2}\sum_i \left(1 + \log\left(\|\tilde{A}_i\|^2 + \tilde{\epsilon}^2\right)\right) \\ &= \frac{1}{2}\|\tilde{B}A\|_F^2 + (d-r)\log\tilde{\epsilon} + \frac{1}{2}\sum_i \left(1 + \log\left(\|\tilde{A}_i\|^2 + \tilde{\epsilon}^2\right)\right) \end{aligned}$$

where in the second line we used the assumption $A = \tilde{A}\tilde{B}A$. Note that for each zero column $\tilde{A}_i = 0$ we have $(1/2)\log(\|\tilde{A}_i\|^2 + \tilde{\epsilon}^2) = \log\tilde{\epsilon}$ so the objective will go to $-\infty$ provided $(d - r + \#\{i : \tilde{A}_i = 0\})\log\tilde{\epsilon} \to -\infty$. Since $\tilde{\epsilon} \to 0$ this is equivalent to asking $d - r + \#\{i : \tilde{A}_i = 0\} > 0$, which is exactly the assumption of the Theorem.

Next we prove the converse direction, i.e. the necessary conditions. Note: we split the first item in the lemma into two conclusions in the proof below (so there are four conclusions instead of three). The first conclusion follows from the first conclusion of Lemma 3.4.1. The second conclusion of Lemma 3.4.1 tells us that

$$0 = \lim_{t\to\infty} \mathbb{E}_{z\sim N(0,I)}\mathbb{E}_{z'\sim N(0,I_{\tilde{r}})}\|Az - \tilde{A}_t(\tilde{B}_tAz + \tilde{D}_t^{1/2}z')\|^2 = \lim_{t\to\infty} \|A - \tilde{A}_t\tilde{B}_tA\|_F^2 + \|\tilde{A}_t\tilde{D}_t^{1/2}\|_F^2$$

which gives us the second and third conclusions above. For the fourth conclusion, since $L_1(\tilde{A}_t, \tilde{B}_t, \tilde{D}_t) \leq L(\tilde{A}_t, \tilde{B}_t, \tilde{D}_t, \tilde{\epsilon}_t)$ we know that $\lim_{t\to\infty} L_1(\tilde{A}_t, \tilde{B}_t, \tilde{D}_t) = -\infty$ and recalling

$$L_1(\tilde{A}, \tilde{B}, \tilde{\epsilon}) = \frac{1}{2\tilde{\epsilon}^2}\|A - \tilde{A}\tilde{B}A\|_F^2 + \frac{1}{2}\|\tilde{B}A\|_F^2 + (d-r)\log\tilde{\epsilon} + \frac{1}{2}\sum_i \left(1 + \log\left(\|\tilde{A}_i\|^2 + \tilde{\epsilon}^2\right)\right)$$

we see that, because the first two terms are nonnegative, this is possible only if the sum of the last two terms goes to $-\infty$. Based on similar reasoning to the sufficiency case, this is only possible if strictly more than $r-d$ of the columns of $(\tilde{A}_t)$ become arbitrarily close to zero; precisely, if there exists δ such that at most $r-d$ of the columns of $\tilde{A}_t$ have norm less than δ, then

$$\begin{aligned}
&(d-r)\log\tilde{\epsilon} + \frac{1}{2}\sum_i \left(1+\log\left(\|\tilde{A}_i\|^2+\tilde{\epsilon}^2\right)\right) \\
&\geq \frac{1}{2}\sum_{i:\|\tilde{A}_i\|\geq\delta} \left(1+\log\left(\|\tilde{A}_i\|^2+\tilde{\epsilon}^2\right)\right) \\
&\geq \frac{1}{2}\sum_{i:\|\tilde{A}_i\|\geq\delta} \left(1+\log\left(\delta^2+\tilde{\epsilon}^2\right)\right)
\end{aligned}$$

which does not go to $-\infty$ as $\tilde{\epsilon}\to 0$ (and the other terms of L_1 are nonnegative). □

3.4.2 Nonlinear VAE

In this section, we give a simple example of a nonlinear VAE architecture which can represent the ground truth distribution perfectly, but has another asymptotic global minimum where it outputs data lying on a manifold of a larger dimension (r^*+s instead of r^* for any $s\geq 1$). The ground truth model is a one-layer network ("sigmoid dataset" in Section 3.6) and the bad decoder we construct outputs a standard Gaussian in r^*+s dimensions padded with zeros. (Note: in the notation of Section 3.6 we are considering a^* with 0/1 entries, but the proof generalizes straightforwardly for arbitrary a^* with the correct support.)

Setup: Suppose $s\geq 1$ is arbitrary and the ground truth $x\in\mathbb{R}^d$ with $d>r^*+s$ is generated in the following way: $(x_1,\ldots,x_{r^*})\sim N(0,I_{r^*})$, $x_{r^*+1}=\sigma(x_1+\cdots+x_{r^*})$ for an arbitrary nonlinearity σ, and $x_{r^*+2}=\cdots=x_d=0$. Furthermore, suppose the architecture for the decoder with latent dimension $r>r^*+1$ is

$$f_{\tilde{A}_1,\tilde{A}_2}(z) := \tilde{A}_1 z + \sigma\left(\tilde{A}_2 z\right)$$

where $\sigma(\cdot)$ is applied as an entrywise nonlinearity, and the encoder is linear, $g(x):=\tilde{B}x$.

Observe that the ground truth decoder can be expressed by taking $\tilde{A}_2$ to have a single nonzero row in position $r+1$ with entries $(1,\ldots,1,0,\ldots,0)$,

$$\tilde{A}_1 = \begin{bmatrix} I_{r^*} & 0 \\ 0 & 0 \end{bmatrix}, \quad \tilde{B} = \begin{bmatrix} I_{r^*} & 0 \\ 0 & 0 \end{bmatrix}.$$

where $\tilde{B}$ is a ground truth encoder which achieves perfect reconstruction.

On the other hand, the following VAE different from the ground truth achieves perfect reconstruction:

$$\tilde{A}_1 = \begin{bmatrix} I_{r^*+s} & 0 \\ 0 & 0 \end{bmatrix}, \quad \tilde{A}_2 = 0, \quad \tilde{B} = \begin{bmatrix} I_{r^*+1} & 0 \\ 0 & 0 \end{bmatrix} \tag{3.9}$$

The output of this decoder is a Gaussian $N\left(0, \begin{bmatrix} I_{r^*+s} & 0 \\ 0 & 0 \end{bmatrix}\right)$, which means it is strictly higher-dimensional than the ground truth dimension r^*. (This also means that if we drew the corresponding plot of to Figure 3.1 (b) for this model, we would get something that looks just like the experimentally obtained result.) We prove in the Appendix that it this is an asymptotic global optima:

Theorem 3.4.6. *Let $s \geq 1$ be arbitrary and the ground truth and VAE architecture is as defined as above. For any sequence $\tilde{\epsilon}_t \to 0$, there exist diagonal matrices $\tilde{D}_t$ such that:*

1. *the VAE loss $L(\tilde{A}_1, \tilde{A}_2, \tilde{B}, \tilde{D}_t, \tilde{\epsilon}_t) \to -\infty$ where $\tilde{A}_1, \tilde{A}_2, \tilde{B}$ are defined by equation 3.9*
2. *The number of coordinates of $\tilde{D}_t$ which go to zero equals $r^* + s$.*

Proof. We show how to pick $\tilde{D}_t$ as a function of $\tilde{\epsilon_t}$ and that if $\tilde{\epsilon_t} \to 0$, the loss goes to $-\infty$. From now on we drop the subscripts.

With these parameters, the VAE loss is

$$\mathbb{E}_{x\sim p^*}\mathbb{E}_{z'\sim N(0,I_r)}\Big[\frac{1}{2\tilde{\epsilon}^2}\|x - f(g(x) + \tilde{D}^{1/2}z')\|^2 + \|g(x)\|^2/2\Big] + d\log(\tilde{\epsilon}) + \mathrm{Tr}(\tilde{D})/2 - \frac{1}{2}\sum_i \log \tilde{D}_{ii}$$

$$= (1/2\tilde{\epsilon}^2)\sum_{i=1}^{r^*+1}\tilde{D}_{ii} + E_{x\sim p^*}\Big[\|x_{1:r^*+1}\|^2/2\Big] + d\log(\tilde{\epsilon}) + \mathrm{Tr}(\tilde{D})/2 - \frac{1}{2}\sum_i \log \tilde{D}_{ii}.$$

Taking the partial derivative with respect to $\tilde{D}_{ii}$ for $i \leq r^* + s$ and optimizing gives $0 = (1/\tilde{\epsilon}^2) + 1 - 1/\tilde{D}_{ii}$ i.e.

$$\tilde{D}_{ii} = \frac{1}{1 + 1/\tilde{\epsilon}^2} = \frac{\tilde{\epsilon}^2}{\tilde{\epsilon}^2 + 1}$$

and plugging this into the objective gives

$$(1/2\tilde{\epsilon}^2)\sum_{i=1}^{r^*+1}\tilde{D}_{ii} + E_{x\sim p^*}\Big[\|x_{1:r^*+1}\|^2/2\Big] + d\log(\tilde{\epsilon}) + \mathrm{Tr}(\tilde{D})/2 - \frac{1}{2}\sum_i \log \tilde{D}_{ii}$$

$$= (1/2)\sum_{i=1}^{r^*+1}\frac{1}{\tilde{\epsilon}^2 + 1} + E_{x\sim p^*}\Big[\|x_{1:r^*+1}\|^2/2\Big] + (d - r^* - s)\log(\tilde{\epsilon})$$

$$+ \mathrm{Tr}(\tilde{D})/2 + \frac{r^*+1}{2}\log(1+\epsilon^2) + \frac{1}{2}\sum_{i=r^*+2}^{r}\log \tilde{D}_{ii}.$$

Setting the remaining $\tilde{D}_{ii}$ to 1, we see that using $d > r^* + s$ that the loss goes to $-\infty$ provided $\tilde{\epsilon} \to 0$, proving the result. □

3.5 Implicit bias of gradient descent in Linear VAE

In this section, we prove that even though the landscape of the VAE loss contains generators with strictly larger support than the ground truth, the gradient flow is *implicitly biased towards low-rank solutions*. We prove this for the simplified loss $L_1(\tilde{A}, \tilde{B}, \tilde{\epsilon}) = \min_{\tilde{D}} L_1(\tilde{A}, \tilde{B}, \tilde{\epsilon}, \tilde{D})$,

which makes the calculations more tractable, though we believe our results should hold for the original loss L as well. The main result we prove is as follows:

Theorem 3.5.1 (Implicit bias of gradient descent)**.** *Let $A : d \times r$ be arbitrary and define W to be the span of the rows of A, let $\tilde{\Theta}(0) = (\tilde{A}(0), \tilde{B}(0), \tilde{\epsilon}(0))$ be an arbitrary initialization and define the gradient flow $\tilde{\Theta}(t) = (\tilde{A}(t), \tilde{B}(t), \tilde{\epsilon}(t))$ by the differential equation equation 3.1. with initial condition $\tilde{\Theta}_0$. If the solution to this equation exists on the time interval $[0, T]$ and satisfies $\max_{t\in[0,T]} \max_j [\|(\tilde{A}_t)_j\|^2 + \tilde{\epsilon}_t^2] \leq K$, then for all $t \in [0, T]$ we have*

$$\sum_{k=\dim(W)+1}^{d} \sigma_k^2(\tilde{A}(t)) \leq \|P_{W^\perp}\tilde{A}^T(t)\|_F^2 \leq e^{-t/K}\|P_{W^\perp}\tilde{A}^T(0)\|_F^2 \tag{3.10}$$

where $P_{W^\perp}$ is the orthogonal projection onto the orthogonal complement of W.

Towards showing the above result, we first show how to reduce to matrices where A has $d - \dim(\text{rowspan}(A))$ rows that are all-zero. To do this, we observe that the linear VAE objective is invariant to arbitrary rotations in the output space (i.e. x-space), so the gradient descent/flow trajectories transform naturally under rotations. Thus, we can "rotate" the ground truth parameters as well as the training parameters.

This is formally captured as Lemma 3.5.2. Recall that by the singular value decomposition $A = USV^T$ for some orthogonal matrices U, V and diagonal matrix S, and rotation invariance in the x-space lets us reduce to analyzing the case where $U = I$, i.e. $A = SV^T$. This matrix has a zero row for every zero singular value.

Lemma 3.5.2 (Rotational Invariance of Gradient Descent on Linear VAE)**.** *Let $L_A(\tilde{A}, \tilde{B}, \tilde{D}, \tilde{\epsilon})$ denote the VAE population loss objective equation 3.4. Then for an arbitrary orthogonal matrix U, we have*

$$L_A(\tilde{A}, \tilde{B}, \tilde{D}, \tilde{\epsilon}) = L_{UA}(U\tilde{A}, \tilde{B}U^T, \tilde{D}, \tilde{\epsilon}).$$

Furthermore,

$$U\nabla_{\tilde{A}} L_A(\tilde{A}, \tilde{B}, \tilde{D}, \tilde{\epsilon}) = \nabla_{U\tilde{A}} L_{UA}(U\tilde{A}, \tilde{B}U^T, \tilde{D}, \tilde{\epsilon})$$

and

$$(\nabla_{\tilde{B}} L_A(\tilde{A}, \tilde{B}, \tilde{D}, \tilde{\epsilon}))U^T = \nabla_{U\tilde{B}} L_{UA}(U\tilde{A}, \tilde{B}U^T, \tilde{D}, \tilde{\epsilon}).$$

As a consequence, if for any $\eta \geq 0$ we define $(\tilde{A}_1, \tilde{B}_1, \tilde{D}_1, \tilde{\epsilon}_1) = (\tilde{A}, \tilde{B}, \tilde{D}, \tilde{\epsilon}) - \eta\nabla L_A(\tilde{A}, \tilde{B}, \tilde{D}, \tilde{\epsilon})$ then

$$(U\tilde{A}_1, \tilde{B}_1U^T, \tilde{D}_1, \tilde{\epsilon}_1) = (U\tilde{A}, \tilde{B}U^T, \tilde{D}, \tilde{\epsilon}) - \eta\nabla_{(U\tilde{A}, \tilde{B}U^T, \tilde{D}, \tilde{\epsilon})} L_{UA}(U\tilde{A}, \tilde{B}U^T, \tilde{D}, \tilde{\epsilon}),$$

i.e. gradient descent preserves rotations by U. The same result holds for the gradient flow (i.e. continuous time gradient descent), or replacing everywhere the loss L by the simplified loss L_1.

Proof of Lemma 3.5.2. We give the proof for L as stated, but it is exactly the same for the simplified loss L_1.

From the objective function equation 3.4 and $U^T = U^{-1}$ observe that

$L_{UA}(U\tilde{A}, \tilde{B}U^T, \tilde{D}, \tilde{\epsilon})$

$$= \frac{1}{2\tilde{\epsilon}^2}\|UA - U\tilde{A}\tilde{B}U^{-1}UA\|_F^2 + \frac{1}{2}\|\tilde{B}U^{-1}UA\|_F^2 + d\log\tilde{\epsilon} + \frac{1}{2}\sum_i \left(\tilde{D}_{ii}\|U\tilde{A}_i\|^2/\tilde{\epsilon}^2 + \tilde{D}_{ii} - \log\tilde{D}_{ii}\right)$$
$$= \frac{1}{2\tilde{\epsilon}^2}\|A - \tilde{A}\tilde{B}A\|_F^2 + \frac{1}{2}\|\tilde{B}A\|_F^2 + d\log\tilde{\epsilon} + \frac{1}{2}\sum_i \left(\tilde{D}_{ii}\|\tilde{A}_i\|^2/\tilde{\epsilon}^2 + \tilde{D}_{ii} - \log\tilde{D}_{ii}\right)$$
$$= L_A(\tilde{A}, \tilde{B}, \tilde{D}, \tilde{\epsilon}).$$

Then from the above and the multivariate chain rule have

$$\nabla_{\tilde{A}} L_A(\tilde{A}, \tilde{B}, \tilde{D}, \tilde{\epsilon}) = \nabla_{\tilde{A}} L_{UA}(U\tilde{A}, \tilde{B}U^T, \tilde{D}, \tilde{\epsilon}) = U^T\left(\nabla_{U\tilde{A}} L_{UA}(U\tilde{A}, \tilde{B}U^{-1}, \tilde{D}, \tilde{\epsilon})\right)$$

so multiplying both sides on the left by U and using $U^T = U^{-1}$ gives the second claim, and similarly

$$\nabla_{\tilde{B}} L_A(\tilde{A}, \tilde{B}, \tilde{D}, \tilde{\epsilon}) = \nabla_{\tilde{B}} L_{UA}(U\tilde{A}, \tilde{B}U^T, \tilde{D}, \tilde{\epsilon}) = (\nabla_{\tilde{B}U^T} L_{UA}(U\tilde{A}, \tilde{B}U^T, \tilde{D}, \tilde{\epsilon}))U$$

gives the third claim. Then the gradient descent claim follows immediately. □

Analysis when A has zero rows. Having reduced our analysis to the case where A has zero rows, the following key lemma shows that for every i such that row i of A (denoted $A^{(i)}$) is zero, the gradient descent step $-\nabla L$ or $-\nabla L_1$ will be negatively correlated with the corresponding row $\tilde{A}^{(i)}$.

Lemma 3.5.3 (Gradient correlation)**.** *If row i of A is zero, then*

$$\sum_{j=1}^r \tilde{A}_{ij}\frac{\partial L}{\partial \tilde{A}_{ij}} \geq \sum_{j=1}^r \tilde{D}_{jj}\tilde{A}_{ij}^2/\tilde{\epsilon}^2, \qquad \sum_{j=1}^r \tilde{A}_{ij}\frac{\partial L_1}{\partial \tilde{A}_{ij}} \geq \sum_{j=1}^r \frac{\tilde{A}_{ij}^2}{\|\tilde{A}_j\|^2 + \tilde{\epsilon}^2}.$$

Proof. First we prove the conclusion for the original loss L. Since $(\tilde{A}\tilde{B}A)_{i\ell} = \sum_{j,k}\tilde{A}_{ij}\tilde{B}_{jk}A_{k\ell}$ we have that

$$\frac{\partial\|A - \tilde{A}\tilde{B}A\|_F^2}{\partial\tilde{A}_{ij}} = \frac{\partial}{\partial\tilde{A}_{ij}}\sum_\ell\left(A_{i\ell} - \sum_{j',k}\tilde{A}_{ij'}\tilde{B}_{j'k}A_{k\ell}\right)^2 = \sum_\ell 2\left(A_{i\ell} - \sum_{j',k}\tilde{A}_{ij'}\tilde{B}_{j'k}A_{k\ell}\right)\left(-\sum_k \tilde{B}_{jk}A_{k\ell}\right)$$

and if we know the corresponding row i in A is zero then this simplifies to

$$\frac{\partial\|A - \tilde{A}\tilde{B}A\|_F^2}{\partial\tilde{A}_{ij}} = \sum_\ell 2\left(\sum_{j',k}\tilde{A}_{ij'}\tilde{B}_{j'k}A_{k\ell}\right)\left(\sum_k \tilde{B}_{jk}A_{k\ell}\right)$$

which means that

$$\sum_j \tilde{A}_{ij}\frac{\partial\|A - \tilde{A}\tilde{B}A\|_F^2}{\partial\tilde{A}_{ij}} = \sum_\ell 2\left(\sum_{j,k}\tilde{A}_{ij}\tilde{B}_{jk}A_{k\ell}\right)^2 = 2\|(\tilde{A}\tilde{B}A)^{(i)}\|^2$$

where the notation $A^{(i)}$ denotes row i of matrix A. Thus, for this term the gradient with respect to row $\tilde{A}^{(i)}$ has nonnegative dot product with row $\tilde{A}^{(i)}$.

Also,

$$\frac{\partial}{\partial \tilde{A}_{ij}}(1/2)\sum_i \tilde{D}_{jj}\|\tilde{A}_j\|^2/\tilde{\epsilon}^2 = \tilde{D}_{jj}\tilde{A}_{ij}/\tilde{\epsilon}^2$$

and so

$$\sum_j \tilde{A}_{ij}\frac{\partial}{\partial \tilde{A}_{ij}}(1/2)\sum_j \tilde{D}_j\|\tilde{A}_j\|^2/\tilde{\epsilon}^2 = \sum_j \tilde{D}_{jj}\tilde{A}_{ij}^2/\tilde{\epsilon}^2$$

which gives the first result.

For the second result with the simplified loss L_1, observe that

$$\frac{\partial}{\partial \tilde{A}_{ij}}\sum_k \log(\|\tilde{A}_k\|^2 + \tilde{\epsilon}^2) = \frac{2\tilde{A}_{ij}}{\|\tilde{A}_j\|^2 + \tilde{\epsilon}^2}$$

so

$$\sum_j \tilde{A}_{ij}\frac{\partial}{\partial \tilde{A}_{ij}}\sum_k \log(\|\tilde{A}_k\|^2 + \tilde{\epsilon}^2) = \sum_j \frac{2\tilde{A}_{ij}^2}{\|\tilde{A}_j\|^2 + \tilde{\epsilon}^2}$$

and the other terms in the loss behave the same in the case of L. Including the factor of $1/2$ from the loss function gives the result. □

The way we use it is to notice that since the negative gradient points towards zero, gradient descent will shrink the size of $\tilde{A}^{(i)}$. Since the size of the matrix $\tilde{A}$ stays bounded, this should mean that for small step sizes the norm of row i of $\tilde{A}$ shrinks by a constant factor at every step of gradient descent on loss L_1. We formalize this in continuous time for the gradient flow, i.e. the limit of gradient descent as step size goes to zero: for the special case of Theorem 3.3.2 in the zero row setting, the corresponding rows of $\tilde{A}$ decay exponentially fast.

Lemma 3.5.4 (Exponential decay of extra rows)**.** *Let A be arbitrary, and let $\tilde{\Theta}(0) = (\tilde{A}(0), \tilde{B}(0), \tilde{\epsilon}(0))$ be an arbitrary initialization and define the gradient flow $\tilde{\Theta}(t) = (\tilde{A}(t), \tilde{B}(t), \tilde{\epsilon}(t))$ to be a solution of the differential equation equation 3.1 with initial condition $\tilde{\Theta}(0)$. If the solution exists on the time interval $[0, T]$ and satisfies $\max_{t\in[0,T]}\max_j[\|(\tilde{A}(t))_j\|^2 + \tilde{\epsilon}(t)^2] \le K$ for some $K > 0$, then for all i such that row i of A is zero we have $\|\tilde{A}^{(i)}(t)\|^2 \le e^{-t/K}\|\tilde{A}^{(i)}(0)\|^2$ for all $t \in [0, T]$.*

Proof. From Lemma 3.5.3 we have that for any such row i,

$$\begin{aligned}\frac{d}{dt}\|\tilde{A}^{(i)}(t)\|^2 &= 2\langle \tilde{A}^{(i)}(t), \frac{d}{dt}\tilde{A}^{(i)}(t)\rangle \\ &= 2\langle \tilde{A}^{(i)}(t), -\nabla_{\tilde{A}(t)^{(i)}} L_1(\Theta_t)\rangle \le -\sum_{j=1}^{r}\frac{(\tilde{A}(t))_{ij}^2}{\|(\tilde{A}(t))_j\|^2 + \tilde{\epsilon}_t^2} \le -(1/K)\|\tilde{A}^{(i)}(t)\|^2\end{aligned}$$

which by Gronwall's inequality implies $\|\tilde{A}^{(i)}(t)\|^2 \le e^{-t/K}\|\tilde{A}^{(i)}(0)\|^2$ as desired. □

Finally, we can use these lemmas to show Theorem 3.5.1.

Proof of Theorem 3.5.1. Before proceeding, we observe that the first inequality in equation 3.10 is a consequence of the general min-max characterization of singular values, see e.g. Horn and Johnson [2012]. We now prove the rest of the statement.

As explained at the beginning of the section, we start by taking the Singular Value Decomposition $A = USV^T$ where S is the diagonal matrix of singular values and U, V are orthogonal. We assume the diagonal matrix S is sorted so that its top-left entry is the largest singular value and its bottom-right is the smallest. Note that this means the first $\dim(W)$ rows of U are an orthonormal basis for W. Note that for any time t, $\|P_{W^\perp}\tilde{A}^T(t)\|_F^2 = \sum_{i=\dim(W)+1}^{d} \|(U\tilde{A}(t)^T)_i\|^2$ because the rows $(U_{\dim(W)+1}, \ldots, U_d)$ are an orthonormal basis for $W^\perp$. Therefore we have that

$$\begin{aligned}\|P_{W^\perp}\tilde{A}^T(t)\|_F^2 &= \sum_{i=\dim(W)+1}^{d} \|(U\tilde{A}(t)^T)_i\|^2 \\ &\leq e^{-t/K} \sum_{i=\dim(W)+1}^{d} \|(U\tilde{A}(0)^T)_i\|^2 = e^{-t/K}\|P_{W^\perp}\tilde{A}^T(0)\|_F^2,\end{aligned}$$

proving the result, provided we justify the middle inequality. Define $A^* := U^T A = SV^T$, which has a zero row for every zero singular value of A, and apply Lemma 3.5.4 (using that the definition of K is invariant to left-multiplication of $\tilde{A}$ by an orthogonal matrix) and Lemma 3.5.2 to conclude that the rows of $U^T\tilde{A}(t)$, i.e. the columns of $U\tilde{A}(t)^T$, corresponding to zero rows of A^* shrink by a factor of $e^{-t/K}$. This directly gives the desired inequality, completing the proof. □

3.6 Simulations

In this section, we provide extensive empirical support for the questions we addressed theoretically. In particular we investigate the kinds of minima VAEs converge to when optimized via gradient descent over the course of training.

Linear VAEs: First, we investigate whether linear VAEs are able to find the correct support for a distribution supported over a linear subspace. The setup is as follows. We choose a ground truth linear transformation matrix A by concatenating an $r^* \times r^*$ matrix consisting of iid standard Gaussian entries with a zero matrix of dimension $(d - r^*) \times r^*$; the data is generated as $Az, z \sim \mathcal{N}(0, I_{r^*})$. Thus the data lies in a r^*-dimensional subspace embedded in a d-dimensional space. We ran the experiment with various choices for r^*, d as well as the latent dimension of the trained decoder (Table 3.1). Every result is the mean over three experiments run with the same dimensionality and setup but a different random seed.

Results: From Table 3.1 we can see that the optima found by gradient descent capture the support of the manifold accurately across all choices of d, r^*, with the correct number of nonzero decoder rows. We also almost always see the correct number of zero dimensions in the diagonal matrix corresponding to the encoder variance.

However, gradient descent is unable to recover the density of the data on the learned manifold in the linear setting — in sharp contrast to the full rank case [Lucas et al., 2019]. We conclude this by comparing the eigenvalues of the data covariance matrix and the learned generator covariance matrix. In order to understand whether the distribution on the linear subspace has the right density, we compute the eigenvalue error by forming matrices $X, \hat{X}$ with n rows, for which

Intrinsic Dimension	3	3	6	6	9	9	12
Ambient Dimension	12	20	12	20	12	20	20
Mean #0's in Encoder Variance	3.3	3.7	6	6	9.3	9	12
Mean # Decoder Rows Nonzero	3	3	6	6	9	9	12
Mean Normalized Eigenvalue Error	0.44	0.71	0.49	0.47	0.30	0.45	0.42

Table 3.1: Optima found by training a linear VAE on data generated by a linear generator (i.e. a linearly transformed standard multivariate gaussian embedded in a larger ambient dimension by padding with zeroes) via gradient descent. The results reflect the predictions of Theorem 3.5.1: the number of nonzero rows of the decoder always match the dimensionality of the input data distribution with no variance while the number of nonzero dimensions of encoder variance is greater than or equal to the nonzero rows. All VAEs are trained with a 20-dimensional latent space. Clearly, the model fails to recover the correct eigenvalues and therefore has a substantially wrong data density function.

each row is sampled from the ground truth and learned generator distribution respectively. We then compute the vector of eigenvalues $\lambda, \hat{\lambda}$ for the ground truth covariance matrix AA^T and empirical covariance matrix $(1/n)\hat{X}^T\hat{X}$ respectively and compute the normalized eigenvalue error $||\hat{\lambda} - \lambda||/||\lambda||$. In no case does the density of the learned distribution come close to the ground truth.

Eigenvalues of Linear Data. As we've discussed, in our linear setting the VAE does not recover the ground truth data density. Since our generative process for ground-truth data is $x = Az$ for a matrix A and z normally distributed, we can characterize the density function by the eigenvalues of the true or estimated covariance matrix. We give figures for the normalized error of these eigenvalues between the learned generator and the ground truth in Table 3.1. A concrete example of eigenvalue mismatch for a problem with 6 nonzero dimensions is a ground-truth set of covariance eigenvalues

$$\lambda = \begin{bmatrix} 0.001 & 0.156 & 1.54 & 5.06 & 9.55 & 16.4 \end{bmatrix}$$

while the trained linear VAE distribution has covariance eigenvalues

$$\hat{\lambda} = \begin{bmatrix} 0.035 & 0.166 & 1.49 & 4.24 & 5.97 & 7.85 \end{bmatrix}.$$

Here, the VAE was easily able to learn the support of the data but clearly is very off when it comes to the structure of the covariances.

Nonlinear Dataset In this section, we investigate whether VAEs are able to find the correct support in nonlinear settings. Unlike the linear setting, there is no "canonical" data distribution suited for a nonlinear VAE, so we explore two setups:

- *Sphere dataset:* The data are generated from the unit sphere concatenated with zero padding at the end. This can be interpreted as a unit sphere embedded in a higher dimensional space. We used 3 layers of 200 hidden units to parameterize our encoder and decoder networks.

To measure how well the VAE has learnt the support of the distribution, we evaluate the average of $(\|\tilde{x}_{:(r^*+1)}\|_2 - 1)^2$, where $\tilde{x}$ are generated by the learnt generator. We will call this quantity *manifold error*. We have also evaluated the *padding error*, which is defined as $\|\tilde{x}_{r^*+2:}\|_2^2$.

- *Sigmoid Dataset:* Let $z \sim \mathcal{N}(0, I_{r^*})$, the sigmoid dataset concatenates z with $\sigma(\langle a^*, z\rangle)$ where $a^* \in \mathbb{R}^{r^*}$ is generated according to $\mathcal{N}(0, I_{r^*})$. We add additional zero paddings to embed the generated data in a higher dimensional ambient space. The decoder is parameterized by a nonlinear function $f(z) = \tilde{A}z + \sigma(\tilde{C}z)$ and the encoder is parameterized by a linear function $g(x) = \tilde{B}x$. The intrinsic dimension of the dataset is r^*.

  To measure how well the VAE has learnt the support of the distribution, we evaluate the average of $(\sigma(\langle a^*, \tilde{x}_{:r^*}\rangle) - \tilde{x}_{r^*+1})^2$, where $\tilde{x}$ are generated by the learnt decoder. We will call this quantity *manifold error*. The *padding error* is defined as similarly as the sphere dataset.

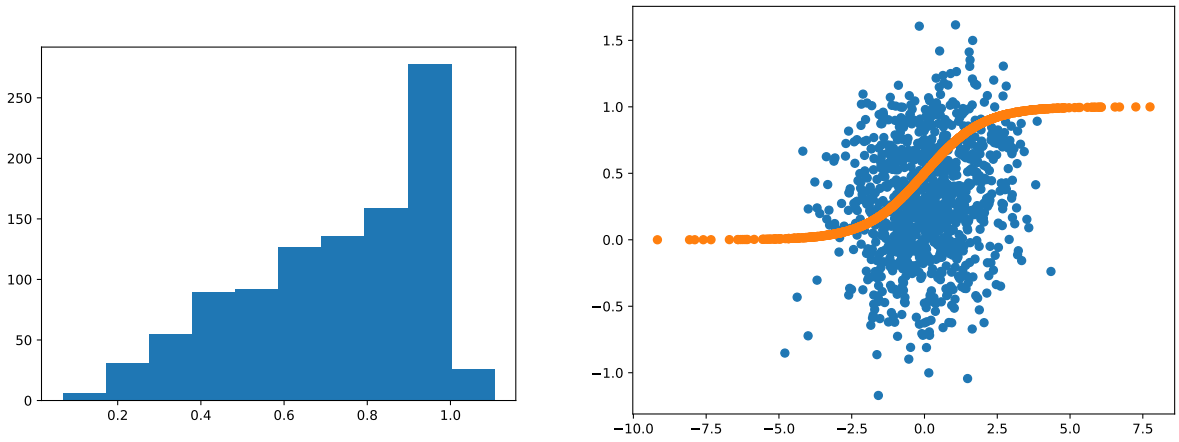


Figure 3.1: A demonstration that in the nonlinear setting (both types of data padded with zeroes to embed in higher ambient dimension, see Setup in Section 3.6) VAE training does not always recover a distribution with the correct support. *Left figure:* A histogram of the norms of samples generated from the VAE restricted to the dimensions which are not zero, which shows many of the points have norm less than 1. (The ground-truth distribution would output only samples of norm 1.) The particular example here is Column 2 in Table 3.3. *Right figure:* Two-dimensional linear projection of data output by VAE generator trained on our sigmoid dataset. The x-axis denotes $\langle a^*, \tilde{x}_{:r^*}\rangle$ and the y-axis is $\tilde{x}_{r^*+1}$, the blue points are from the trained VAE and the orange points are from the ground truth. In contrast to the ground truth data, which satisfies the sigmoidal constraint $x_{r^*+1} = \sigma(\langle a^*, x_{:r^*}\rangle)$, the trained VAE points do not and instead resemble a standard gaussian distribution. This is a case that closely resembles the example provided in Theorem 3.4.6. Also similar to Theorem 3.4.6, the VAE model plotted here (from Column 6 in Table 3.2) achieves nearly-perfect reconstruction error, approximately 0.001.

Results: In both of the nonlinear dataset experiments, we see that the number of zero entries in the diagonal encoder variance is less reflective of the intrinsic dimension of the manifold than the linear dataset. It is, however, at least as large as the intrinsic dimension (Table 3.3, 3.2). We consider a coordinate to be 0 if it's less than 0.1. We found that the magnitude of each coordinate

Intrinsic Dimensions	3	3	5	5	7	7
Ambient Dimensions	7	17	11	22	15	28
VAE Latent Dimensions	6	8	10	16	13	24
Mean Manifold Error	0.09	0.13	0.23	0.24	0.18	0.28
Mean #0's in Encoder Variance	3	3.6	6	6.3	7.3	8

Table 3.2: Optima found by training a VAE on the sigmoid dataset. The VAE training consistently yields encoder variances with number of 0 entries greater than or equal to the intrinsic dimension.

to be well separated, i.e. the smaller coordinates tend to be smaller than 0.1 and the larger tend to be bigger than 0.5. Thus the threshold selection is not crucial. We did not include padding error in the tables because it reaches zero in all experiments

We show the progression of manifold error, decoder variance and VAE loss during training for the sphere data in Figure 3.3 and for the sigmoid data in Figure 3.2. Datasets of all configurations of dimensions reached close to zero decoder variances, meaning the VAE loss is approaching $-\infty$. To demonstrate Theorem 3.4.6, we took examples from both datasets to visualize their output.

For the sphere dataset, we visualize the data generated from the model, with 8 latent dimensions, trained on unit sphere with 2 intrinsic dimensions and 16 ambient dimensions (Column 2 in Table 3.3). Its training progression is shown as the orange curve in Figure 3.3 . We create a histogram of the norm of its first 3 dimensions (Figure 3.1 (a)) and found that more than half of the generated data falls inside of the unit sphere. The generated data has one intrinsic dimension higher than its training data, despite its decoder variance approaching zero, which is equivalent to its reconstruction error approaching zero by Lemma 3.4.2.

In the sigmoid dataset, the featured model has 24 latent dimension, and is trained on a 7-dimensional manifold embedded in a 28-dimensional ambient space. We produced a scatter plot given 1000 generated data points $\tilde{x}$ from the decoder. The x-axis in the Figure 3.1(b) is $\langle a^*, \tilde{x}_{:r^*} \rangle$ and the y-axis is $\tilde{x}_{r^*+1}$. In contrast to the groundtruth data, whose scatter points roughly form a sigmoid function, the scatter points of the generate data resemble a gaussian distribution. This closely resembles the example provided in Theorem 3.4.6. Hence, despite its decoder variance and reconstruction error both approaching zero and loss consistently decreasing, the generated data do not recover the training data distribution and the data distribution recovered has higher intrinsic dimensions than the training data. We also investigated the effect of lower bounding the decoder variance as a possible way to improve the VAE performance (details are given in Appendix 3.7). This enabled the VAE to recover the correct manifold dimension in the sigmoid example, but not the sphere example; methods of improvements to the VAE's manifold recovery is an important direction for future work.

3.7 Experiments with Decoder Variance Clipping

As was suggested by an anonymous reviewer, one potential way to evade the results in our paper is to restrict the decoder variance from converging to 0. In this section, we examine (empirically) the impact of clipping the decoder variance during training. We caveat though, that our paper

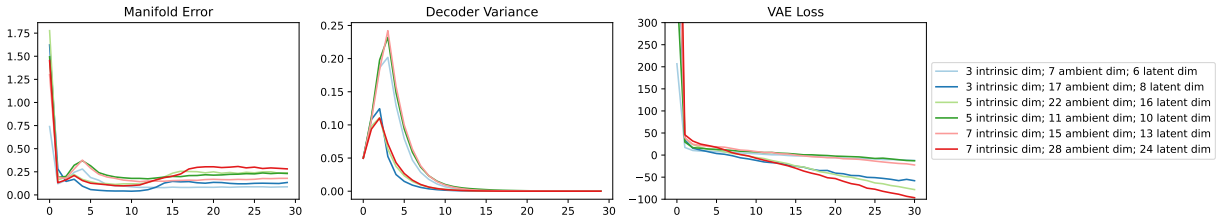


Figure 3.2: VAE training on 6 datasets with different choices of dimensions for sigmoidal dataset (see Setup in Section 3.6). The x-axis represents every 5000 gradient updates during training. The left-most figure is the manifold error (see Setup in Section 3.6), The middle and right figure confirms that the decoder variance approaches zero and the VAE loss is steadily decreasing during the finite training time.

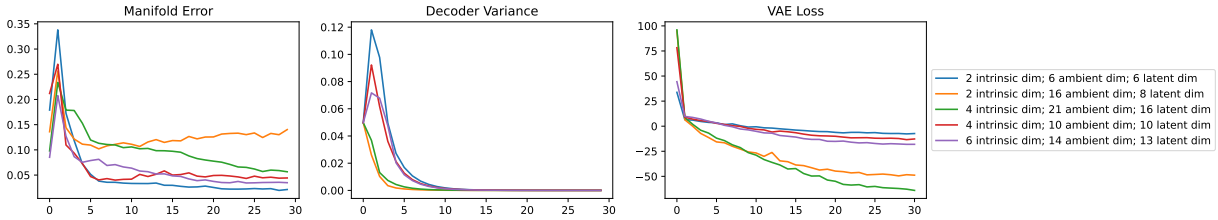


Figure 3.3: VAE training on 5 datasets generated by appending zeros to uniformly random samples from a unit sphere to embed in a higher dimensional ambient space. The x-axis represents each iteration of every 5000 gradient updates. The left-most figure is the manifold error (see Setup in Section 3.6), The middle and right figure confirms that the decoder variance approaches zero and the VAE loss is steadily decreasing during the finite training time.

Intrinsic Dimensions	2	2	4	4	6
Ambient Dimensions	6	16	10	21	14
VAE Latent Dimensions	6	8	10	16	13
Mean Manifold Error	0.02	0.14	0.04	0.06	0.03
Mean #0's in Encoder Variance	3	5	5	6	7

Table 3.3: Optima found by training a VAE on data generated by padding uniformly random samples from a unit r^*-sphere with zeroes, so that the sphere is embedded in a higher ambient dimension. We evaluated the manifold error as described in the setup. The VAE training on this dataset has consistently yielded encoder variances with number of 0 entries greater than the number of intrinsic dimension.

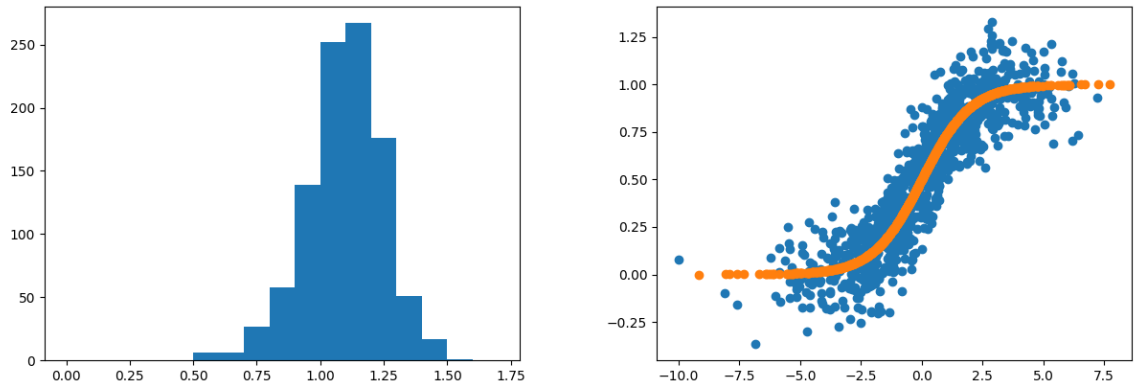


Figure 3.4: A demonstration of how the data points generated by the model trained with clipped decoder variance is distributed. *Left figure:* A histogram of the norms of samples generated from the VAE restricted to the dimensions which are not zero, which shows many of the points have norm less than 1. (The ground-truth distribution would output only samples of norm 1.) The particular example here is Column 2 in Table 3.5. The data points that do not fall on the sphere tend to lie on both sides of it whereas the those generated without decoder variance clipping tend to lie inside the sphere as in Figure 3.1. *Right figure:* Two-dimensional linear projection of data output by VAE generator trained on our sigmoid dataset. The x-axis denotes $\langle a^*, \tilde{x}_{:r^*} \rangle$ and the y-axis is $\tilde{x}_{r^*+1}$, the blue points are from the trained VAE and the orange points are from the ground truth. The generated data points roughly capture the shape of the sigmoid function.

does not analyze the landscape of the resulting constrained optimization problem, so our results don't imply anything about this regime.

We conduct the same nonlinear experiments described in Section 3.6 where we fit VAEs to data generated from spheres and linear sigmoid functions. The only change is to clip the decoder variance when it goes below a certain threshold. In the figures below, the featured threshold is $e^{-4} \approx 0.018$, though we tried also e^{-2}, e^{-3}, e^{-5}, e^{-6}, and e^{-8} with similar outcomes. We initialize the decoder variance with e^{-3} for this set of experiments, so the optimization still can decrease it.

With this change, the optimization process on the sigmoid dataset does yield encoder variances with their number of zeros reflective of their intrinsic dimensions as in Table 3.4. For the sphere experiment, this still does not happen, as in Table 3.5. In fact, the model consistently recovers one more dimension than the true intrinsic dimension of the manifold and the smaller encoder variances can be as large as 0.1. We also provide a figure (Figure 3.4) in the same style as Figure 3.1. We see that training with a clipped decoder variance of e^{-4} allows the model to better capture the general shape of the sigmoid function than training without clipping, though the variance of the generated points is high for both of the sphere and sigmoid datasets. Additional experiments with more thresholds are in Figure 3.7 and Figure 3.8. As we decrease the threshold from e^{-4} to e^{-8}, the fit of the data points concentrate closer to the groundtruth data before getting further away; at e^{-8}, the data distributions for both dataset resemble the unclipped experiments again. Other training details, such as the general trend of manifold error, encoder variance and VAE loss, can be referred to in Figure 3.5 and 3.6.

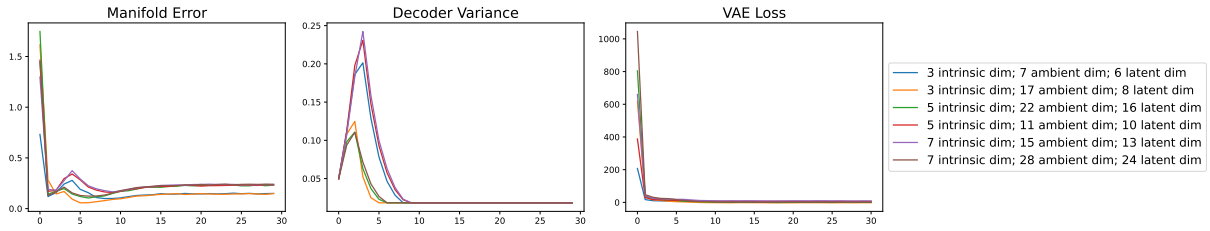


Figure 3.5: VAE training on 6 datasets with different choices of dimensions for sigmoidal dataset (see Setup in Section 3.6). The x-axis represents every 5000 gradient updates during training. The left-most figure is the manifold error (see Setup in Section 3.6), The middle and right figure shows that as the decoder variance is bounded below, the VAE loss stops decreasing further.

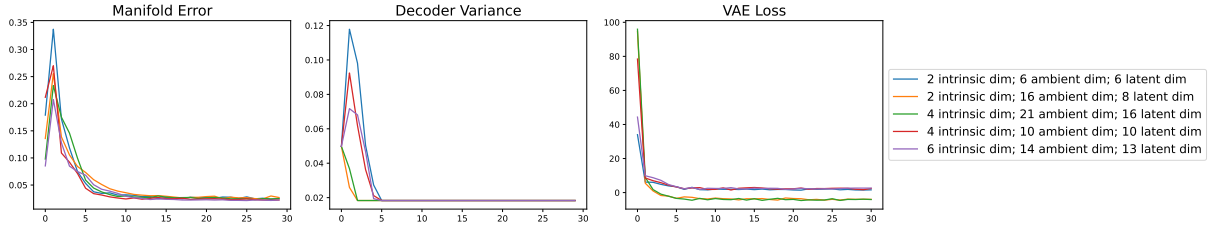


Figure 3.6: VAE training on 5 datasets generated by appending zeros to uniformly random samples from a unit sphere to embed in a higher dimensional ambient space. The x-axis represents each iteration of every 5000 gradient updates. The left-most figure is the manifold error (see Setup in Section 3.6), The middle and right figure shows that as the decoder variance is bounded below, the VAE loss stops decreasing further.

Overall, the benefit of clipping the decoder variance during training is inconclusive as we see inconsistent results in the sphere and sigmoid datasets. Designing more algorithms to improve the ability of VAE's to recover data supported on a low dimensional manifold is an important direction for future work—both empirical and theoretical.

Intrinsic Dimensions	3	3	5	5	7	7
Ambient Dimensions	7	17	11	22	15	28
VAE Latent Dimensions	6	8	10	16	13	24
Mean Manifold Error	0.15	0.15	0.23	0.23	0.24	0.24
Mean #0's in Encoder Variance	3	3	5	5	7	7

Table 3.4: Optima found by training a VAE on the sigmoid dataset. The VAE training yields encoder variances with number of 0 entries equal to the intrinsic dimension.

Intrinsic Dimensions	2	2	4	4	6
Ambient Dimensions	6	16	10	21	14
VAE Latent Dimensions	6	8	10	16	13
Mean Manifold Error	0.03	0.03	0.03	0.02	0.02
Mean #0.1's in Encoder Variance	3	3	5	5	7

Table 3.5: Optima found by training a VAE on data generated by padding uniformly random samples from a unit r^*-sphere with zeroes, so that the sphere is embedded in a higher ambient dimension. We evaluated the manifold error as described in the setup. The VAE training on this dataset has consistently yielded encoder variances with number of 0.1 entries greater than the number of intrinsic dimension.

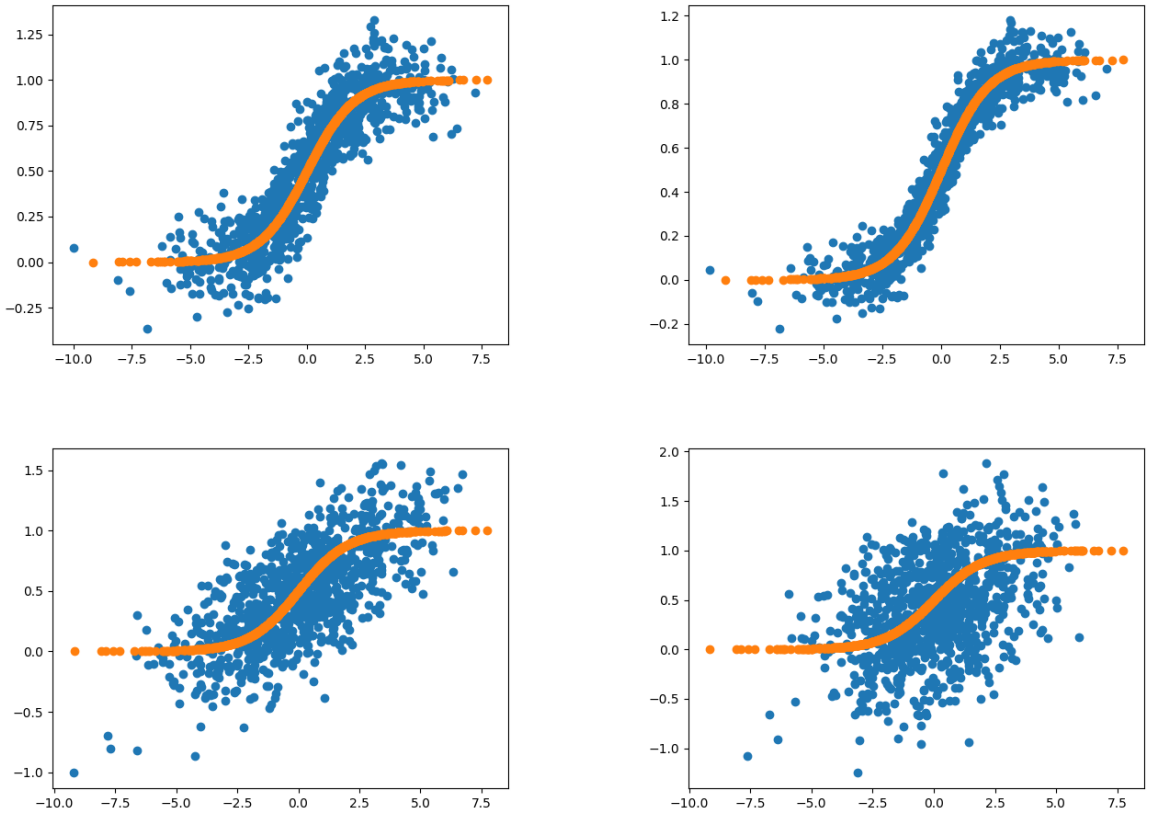


Figure 3.7: From top left to right bottom are scattered points generated in the same way as in Figure 3.4(right) with the clipping threshold set at e^{-4}, e^{-5}, e^{-6} and e^{-8}. We notice that the scattered points were able to capture the sigmoidal shape with a threshold at e^{-4} and e^{-5}. But as the threshold lowers further, the resemblance disappears. Between e^{-4} and e^{-5}, it is clear that the smaller threshold leads to a scatter plot more concentrated around the sigmoid function.

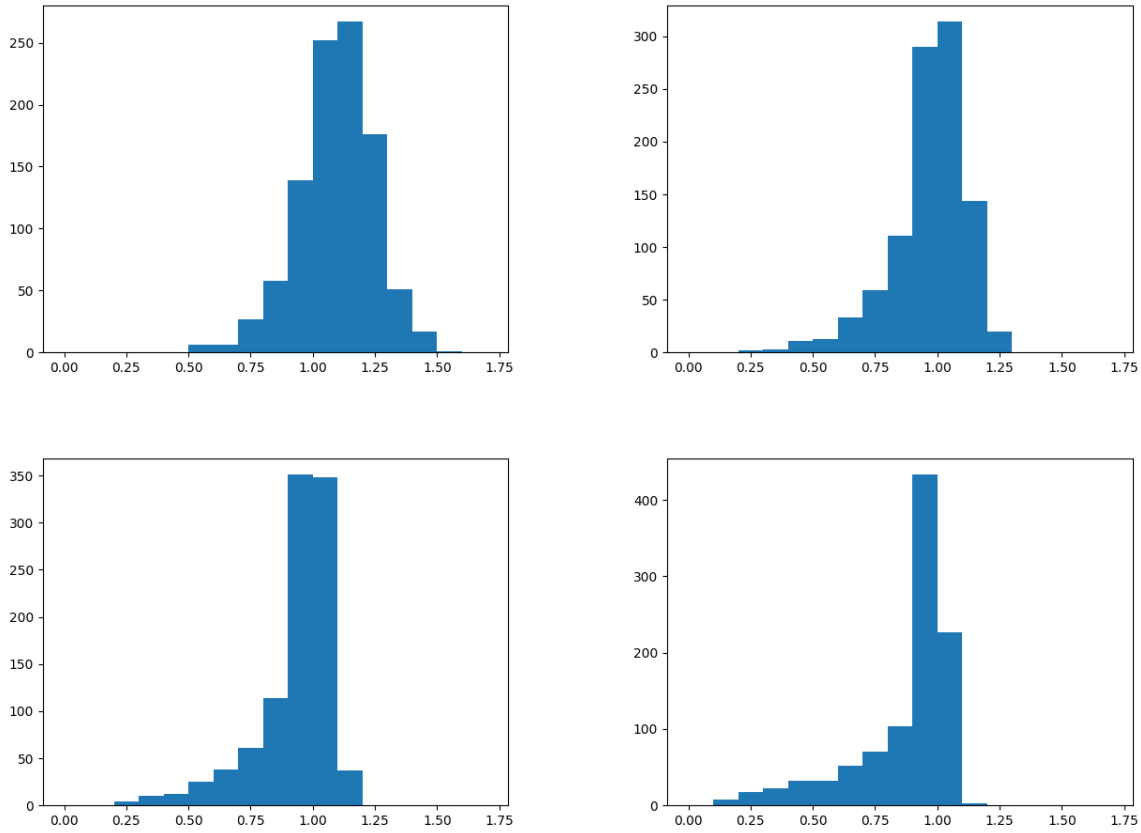


Figure 3.8: From left to right are histograms of generated points' distance to the center of the sphere with clipping threshold set at e^{-4}, e^{-5}, e^{-6} and e^{-8}. As the threshold lowers, the number of points with a distance larger than 1 decreases, but the points inside the sphere reach closer to center.

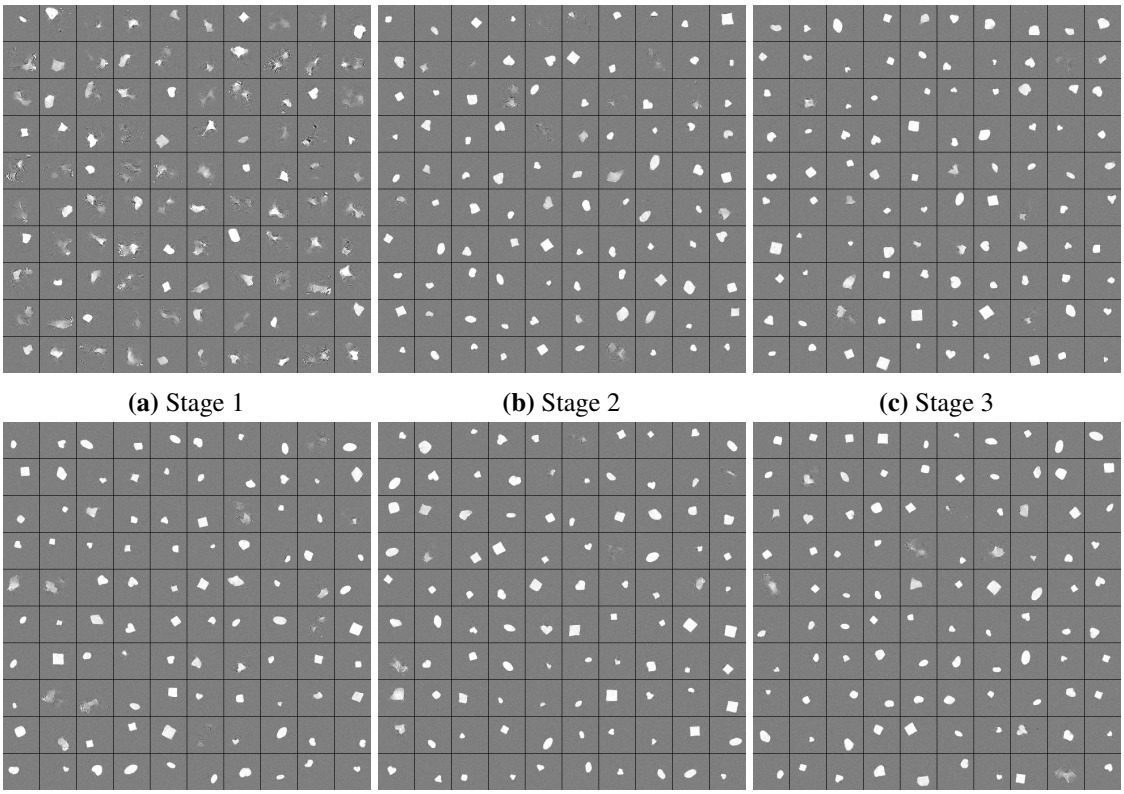
3.8 Additional Experiments with Multi-Stage VAE

In this section, we show the effect of multi-stage VAE's on manifold recovery and data density recovery.

The multi-stage VAE is trained with the latent variable of the previous stage to be the input of the VAE in the next stage. The decoder variance is set to be tunable in all stages. During sampling, the latent variable of the VAE in the last stage is sampled from a standard normal, the output of the decoder of in one stage of VAE becomes the latent for the next stage.

We studied the effect of multi-stage VAE's effect on manifold recovery on sphere dataset in Figure 3.13 as described in the experiment section and mnist dataset in Figure 4.4 LeCun et al. [1998], rotated mnist dataset in Figure 3.12, as well as a dsprite dataset Matthey et al. [2017] in Figure 3.9. We demonstrate the output from VAE training from each stage in Figure. The observation we made from these experiments is that, the manifold recovery improves significantly on the second training stage, but the performance gain plateaus afterwards.

We perform additional experiments on density recovery on the sphere dataset Figure 3.11.



(a) Stage 1 **(b)** Stage 2 **(c)** Stage 3

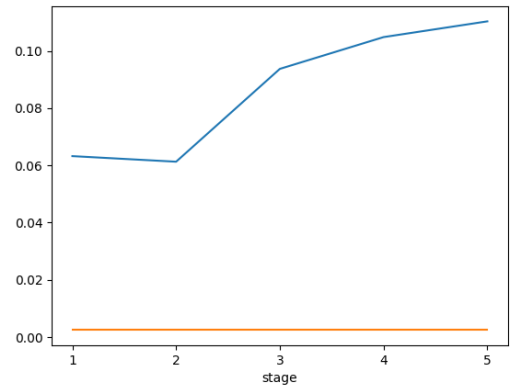
(d) Stage 4 **(e)** Stage 5 **(f)** Stage 6

Figure 3.9: Example outputs of the generator from learning the dsprites dataset.

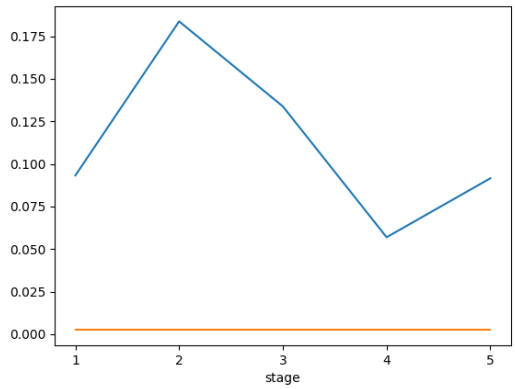
(a) Stage 1 **(b)** Stage 2 **(c)** Stage 3

(d) Stage 4 **(e)** Stage 5 **(f)** Stage 6

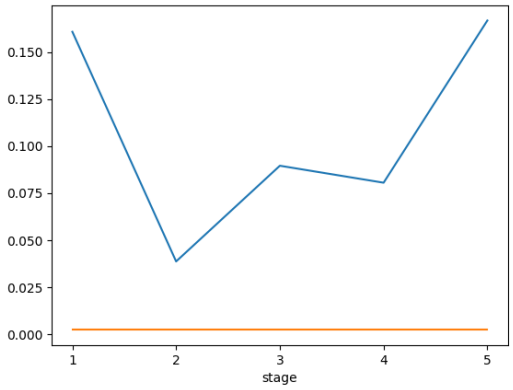
Figure 3.10: Example outputs of the generator from learning the mnist dataset across 6 stages.



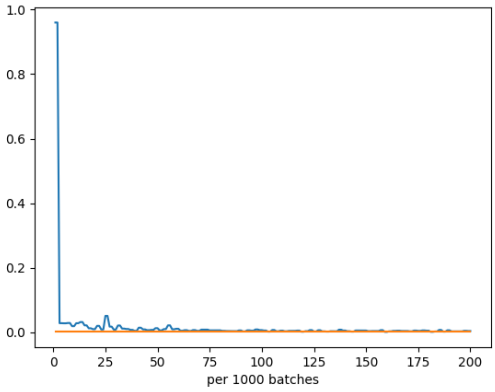
(a) VAE #1



(b) VAE #2

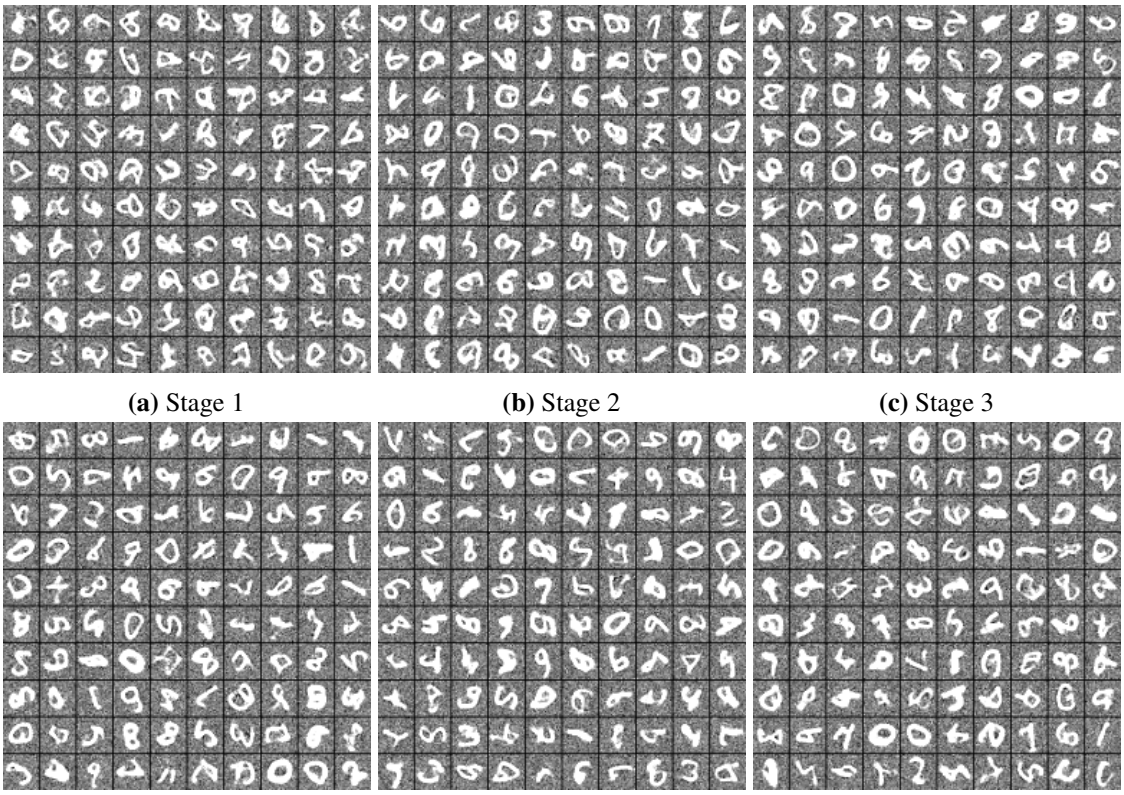


(c) VAE #3



(d) Supervised Generator

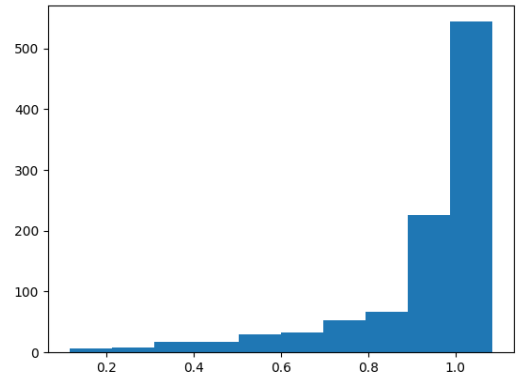
Figure 3.11: The sample covariance difference. VAE #1, #2 and #3 are trained on 3 dimensional unit sphere with padding dimension of 3, 5, and 7 respectively. The x-axis represents the stage number. the yellow line represents the baseline of the sample covariance difference by taking the norm of the difference of two ground-truth sample covariances. The supervised generator takes on the exact same architecture of the VAE decoder. Its input and output are the latent and output of the ground-truth generator. During training, the sample covariance difference eventually reach the baseline level.



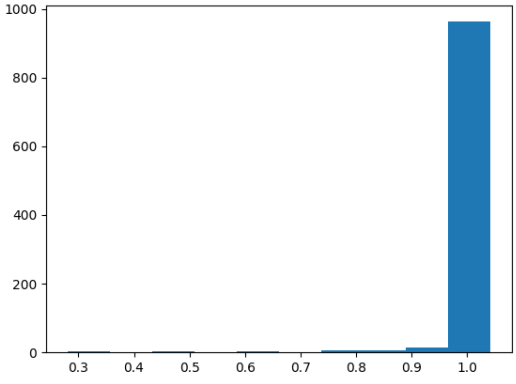
(a) Stage 1 **(b)** Stage 2 **(c)** Stage 3

(d) Stage 4 **(e)** Stage 5 **(f)** Stage 6

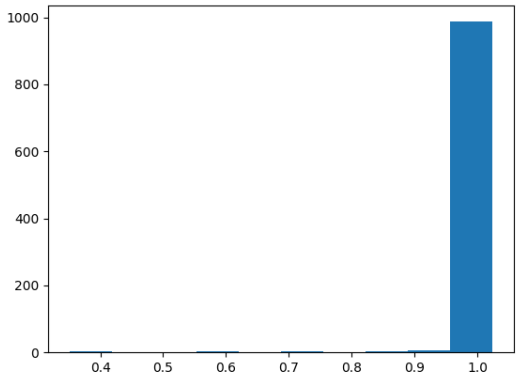
Figure 3.12: Example outputs of the generator from learning the rotated mnist dataset across 6 stages.



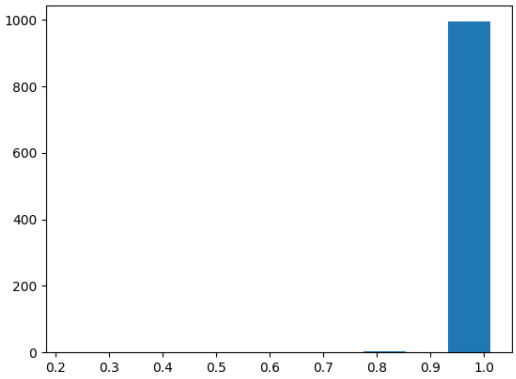
(a) Stage 1



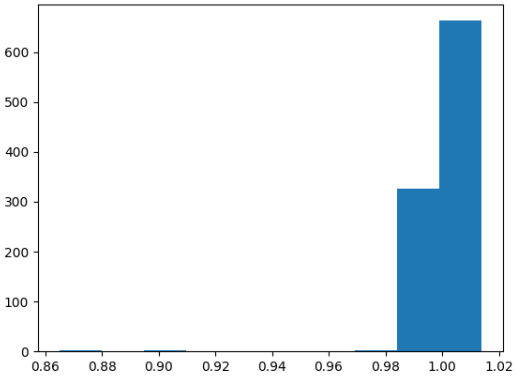
(b) Stage 2



(c) Stage 3



(d) Stage 4



(e) Stage 5

Figure 3.13: A bar plot on the distribution of the generated data points' norms. This model is trained on data generated from a 2-dimensional unit sphere with 7 dimensional padding.

For the experiments, the data is generated from a 2-dimensional unit sphere padded with 3, 5, and 7-dimensional zero vectors respectively for three sets of datasets. We calculated the sample covariance from the decoder during the generation process, and compared against the ground-truth data's covariance. We calculate the norm of the difference between the sample covariance and the ground-truth data's covariance as a metric for density recover. We also sample ground-truth data twice to calculate two separate covariances, the norm of their difference is used as a baseline. We found that none of the stages can recover the ground-truth sample covariance. And more stages do not lead to better density recovery. To verify that the architecture is capable of recovering the density of the dataset, we construct a supervised model with the same architecture as the VAE decoder (4 layers of 500 neurons per layer). The input and output pair for the supervised model is the latent variable sampled from the standard normal distribution and the VAE training data which is normalized and padded from the latent variable. The result shows that the supervised model is capable of eventually recovering the sample covariance as the sample covariance difference approaches the baseline in Figure 3.11d

3.8.1 Conclusion

In this section, we investigated a one-stage VAE's ability to recover data manifold and density. We conducted synthetic experiments on linear and non-linear data and found that it is able to recover the data manifold of linear data due to an implicit bias towards a low-rank solution but it is not able to recover data density. On non-linear data, it is not guaranteed to recover the data manifold or density. This has significant implications for the quality of the generated data as non-linear data encompasses many data types in real life, such as images, languages, and molecular data. In Figure 4.4, we can see the model trained on MNIST data is blurrier and harder to identify each number than the ground-truth image. In the next section, we will present the application of VAE in drug discovery and that poor recovery of data can lead to discrepancies in properties with the training data among the generated data. This highlights the importance of improving the manifold recovery ability of VAEs. Thus in the last two sections, we explored decoder variance clipping and multi-stage VAE for improved manifold recovery. We found that decoder variance clipping does not achieve satisfactory improvements in manifold recovery and training additional stages does improve manifold recovery. However, more than 2 stages of VAE do not necessarily induce substantial improvements. The multi-stage VAE training scheme also does not lead to data density recovery, despite the fact that the decoder architecture is shown to be capable of density recovery.

Chapter 4

Improving Molecule Generation via Multi-Stage VAE

4.1 Introduction

The use of generative models in the domain of drug discovery has recently seen rapid progress. These methods can leverage large-scale molecule archives to synthesize novel molecules with similar properties as potential candidates for future drugs [Duvenaud et al., 2015, Liu et al., 2018a, Segler et al., 2018, You et al., 2018, Jin et al., 2018, 2020a, Polykovskiy et al., 2020, Jin et al., 2020b, Satorras et al., 2021, Maziarz et al., 2021, Hoogeboom et al., 2022]. Molecules can be represented in Simplified Molecular Input Line Entry System (SMILES) format [Weininger, 1988] and as molecular graphs [Bonchev, 1991]. Graph neural networks (GNNs) can take into account atoms, bonds, and other structural information in molecular graph representations while conventional sequence models (e.g., RNNs) are more compatible with SMILES. Achieving structural validity is the first step to AI-driven drug discovery. To do so, GNN methods [Liu et al., 2018a, Simonovsky and Komodakis, 2018, Jin et al., 2020a, Maziarz et al., 2021] can constrain the output space based on chemical rules and SMILES-based approaches [Gómez-Bombarelli et al., 2018, Blaschke et al., 2018] benefit from large molecular data.

Besides structural validity, other chemical properties such as drug-likeness (QED) [Bickerton et al., 2012], Synthetic Accessibility (SA) [Ertl and Schuffenhauer, 2009], LogP (The Octanol-Water Partition Coefficient) [Wildman and Crippen, 1999] and molecular weight (MW) are also critical factors determining whether candidate molecules can be synthesized in a laboratory or be effective in real-world applications. A molecule's activity level on protein targets, whether to inhibit or to activate, is another very important property when treating specific diseases. A molecule that interacts successfully with the protein target is considered active and an activity score is measured by quantifying its ability to either activate or inhibit the protein target's biological function. Researchers collect and curate large molecule datasets, such as ChEMBL [Mendez et al., 2019] and ZINC [Irwin and Shoichet, 2005], that contain an array of bioactive molecules together with information about their properties and protein targets. By training generative models on a curated set of molecules, the models can learn how to generate new molecules that are similar in properties to those in the training set and potentially be able to produce novel drug candidates

that satisfy multiple objectives, e.g. being drug-like and active against multiple protein targets. Benchmark metrics [Polykovskiy et al., 2020, Brown et al., 2019] are created to measure how similar the generated molecules are to the target dataset structurally and property-wise. The state-of-the-art results, however, show that there is still room for improvements. Multi-objective generation by incorporating property predictors in the training pipeline [Jin et al., 2020b, Maziarz et al., 2021] is a promising avenue to address this problem, but there are also potential drawbacks. As Winter et al. [2019] summarized, in drug discovery, the optimization objectives can be complex, conflicting, ill-defined or evolving over time.

In this paper, we introduce an objective-agnostic and easy-to-implement technique to improve existing VAE-based molecule generation models – training subsequent stages of VAE's to generate latent variables for the preceding-stage VAE. To illustrate how this approach can enhance the manifold recovery of VAE models, we first study a simple MLP model trained on a synthetic sphere dataset. We then evaluate our method in an unconstrained molecule generation task and a fine-tuning task. In these experiments, we demonstrate the following claims:

- The multi-stage VAE is able to bring the properties of the generated molecules *closer in distribution* to the test set when training on the ChEMBL dataset;
- Fine-tuning the multi-stage VAE on the curated active molecules of two protein targets results in *more* active outputs than fine-tuning only the first-stage model;
- In both tasks, our method can achieve comparable or better results than specialized methods that introduce property predictors into the training pipeline to optimize one or multiple target objectives.

4.2 Related Work

Most prior works fall, based on employed molecule representation, into one of the following families – namely, the SMILES string approach, the molecular graph approach, and the 3D point set approach. Many methods have been proposed to generate molecules as SMILES strings [Segler et al., 2018, Gómez-Bombarelli et al., 2018]. Kusner et al. [2017] and Dai et al. [2018] leverage the syntax of the SMILES format to constrained the output of the VAE model and improve validity of the generated molecules. GANs have also been proposed to generate SMILES [Kadurin et al., 2017, Prykhodko et al., 2019, Guimaraes et al., 2017]. We include Prykhodko et al. [2019]'s work as a baseline in the experiment (Section 4.4) but its performance on property statistics is worse than the VAE approaches. Molecular graphs carry more information about molecular structure than SMILES, and GNNs can effectively incorporate the additional information into the learning process [Duvenaud et al., 2015, Liu et al., 2018a, Jin et al., 2018, Maziarz et al., 2021]. Jin et al. [2020a] proposed to generate via substructures in a coarse-to-fine manner to adapt to larger molecules, such as polymers. 3D representations of molecules are gaining traction in the research community as they capture detailed spatial information of the molecules and autoregressive models [Gebauer et al., 2019, 2022, Luo et al., 2021] and diffusion models [Hoogeboom et al., 2022, Xu et al., 2022, Vignac et al., 2022] are common approaches to make use of this type of representation. However, work in diffusion models so far has focused on generation of smaller molecules such as in QM9 Ramakrishnan et al. [2014]. Our paper focuses on improving upon the

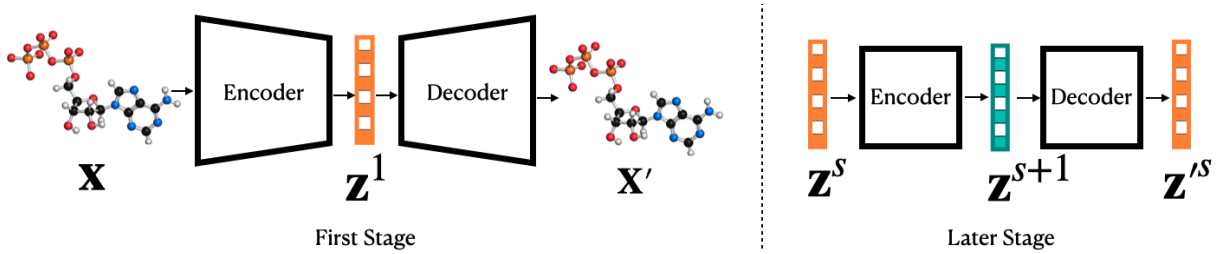


Figure 4.1: Overview of multi-stage VAE. In the first stage, the VAE trains on the molecule data $\boldsymbol{x}_i$ and obtains the latent variables $\boldsymbol{z}_i^1$ from $\boldsymbol{x}_i$. The later-stage VAE is trained on the latent variables of the preceding-stage VAE. $\boldsymbol{z}_i^s$'s from the s-th stage VAE become the input to the $s+1$-th stage VAE during training. The later-stage VAE's input dimension is equal to the output dimension. During sampling, we sample $\mathbf{z} \sim \mathcal{N}(0, I)$ and obtain $\boldsymbol{z}_i'^s$ from the decoder. The output of a subsequent-stage VAE decoder is used as the input for the preceding-stage VAE decoder until the latent variable is decoded into a new molecule $\boldsymbol{x}_i'$ in the first-stage VAE.

existing *VAE-based* approaches.

4.3 Method

The VAE [Kingma and Welling, 2013] has enabled great success in the image generation domain and more recently it is also adopted for molecule generation. Prior work [Kusner et al., 2017, Dai et al., 2018, Jin et al., 2019] has focused on perfecting underlying architectures, e.g. improving generation of large molecules or enforcing syntax on SMILES output. However, perfecting architectures does not necessarily improve the molecule's properties, which could be a result of VAE's inherent learning deficiency in manifold recovery [Dai and Wipf, 2019, Koehler et al., 2021]. The *manifold hypothesis* [Fefferman et al., 2016, Goodfellow et al., 2016] states that many high-dimensional real life data lie on low-dimensional manifolds embedded in high-dimensional ambient spaces. Koehler et al. [2021] found that the VAE is not guaranteed to recover the low-dimensional manifold where a nonlinear dataset lie. We show that a multi-stage VAE method can improve manifold recovery as demonstrated in a synthetic experiment (Figure 4.2), and further, can enhance the performance of pre-existing VAE models.

4.3.1 Variational Autoencoder

The variational inference framework assumes that the data $\mathbf{x} \in \mathbb{R}^d$ is generated from a latent variable $\mathbf{z} \in \mathbb{R}^r$ via a nonlinear transformation. In VAEs, the prior distribution $p(\mathbf{z}) = \mathcal{N}(0, I_r)$. The VAE model consists of a tractable encoder $q_\phi(\mathbf{z}|\mathbf{x})$ and a decoder $p_\theta(\mathbf{x}|\mathbf{z})$. Here ϕ is the so-called variational parameter, and θ denotes the generative model parameters. The VAE model seeks to maximize the likelihood of the data, denoted as $\log p_\theta(\mathbf{x}) = \log \int p(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z}$. However, in this model the marginalization is intractable due to the inherent complexity of the generator, or the decoder, thus an approximation of the objective is needed. In VAE, the encoder and the decoder work together to approximate a lower bound to the likelihood of the data. By optimizing this lower bound we aim to increase the likelihood objective. This approximation, called variational

approximation, enables efficient posterior inference of the latent variable $\mathbf{z} \in \mathbb{R}^r$ given input $\boldsymbol{x}_i$ and for marginal inference of the output variable $\mathbf{x} \in \mathbb{R}^d$. The VAE objective function consists of a KL divergence term D_{KL} and a reconstruction term:

$$\begin{aligned}\mathcal{L}(\theta, \phi; \boldsymbol{x}) = &- D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p(\mathbf{z})) \\ &+ \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] \leq \log p_\theta(\boldsymbol{x})\end{aligned} \tag{4.1}$$

The encoder and the decoder are parameterized as:

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_\phi, \Sigma_\phi) \tag{4.2}$$

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}_\theta, \sigma_\theta^2 I_d) \tag{4.3}$$

4.3.2 Multi-Stage VAE

Dai and Wipf [2019] argued that training a continuous VAE with a fixed decoder variance, e.g. ${\sigma_\theta}^2 = 1$[1], could add unnecessary noise to the output. While training a VAE with a tunable decoder variance, they observed that the decoder variance σ_θ^2 tends to approach zero as the loss, $-\mathcal{L}(\theta, \phi; x) = \text{Eq } (4.4) + \text{Eq } (4.5)$ in the below equations, approaches negative infinity. However, a second VAE trained on the encoded latent variables of the first VAE yielded crisper and more realistic images than the stage-1 VAE. They *hypothesized* that the data manifold is recovered in the first stage and the density is recovered in the second stage.

$$-\log p_\theta(\mathbf{x}|\mathbf{z}) \propto \frac{\|\mathbf{x} - \boldsymbol{\mu}_\theta\|_2^2}{\sigma_\theta^2} + d \log \sigma_\theta^2 \tag{4.4}$$

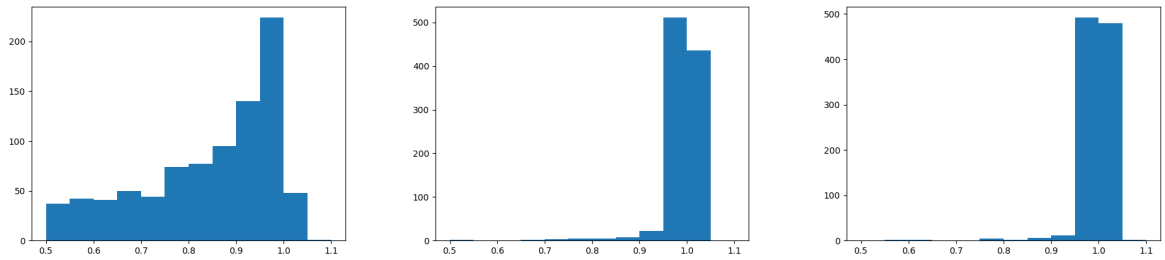
$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p(\mathbf{z})) \propto \text{trace}(\Sigma_\phi) + \|\boldsymbol{\mu}_\phi\|_2^2 - \log|\Sigma_\phi| \tag{4.5}$$

Koehler et al. [2021] tested this hypothesis on linear and nonlinear data. The landscape analysis of Eq (4.4) + Eq (4.5) is tractable with linear data, which is defined as $x = Az$ where $A \in \mathbb{R}^{d \times r}$ and $z \sim \mathcal{N}(0, I_r)$. They found that the optimum of the loss function can have a higher rank than the rank of A but the eventual recovery of the linear manifold is due to an implicit bias towards lower-ranked solutions in the training dynamics. Synthetic experiments on accessible manifolds were conducted in place of a general loss landscape analysis for non-linear data due to the complexity of loss functions with added nonlinearity (e.g., when encoder and decoder are each parameterized with an MLP). The experiments revealed that the manifold of nonlinear data is *not* guaranteed to be recovered by a one-stage VAE.

We extend the synthetic experiments conducted by Koehler et al. [2021] and demonstrate in the following section that a multi-stage VAE can *improve* manifold recovery on non-linear data, providing insights to the phenomenon observed in Dai and Wipf [2019].

Our main contribution is that we apply the multi-stage VAE to the challenging task of molecule generation. To overcome the difficulties in this task, we use complex neural network architectures such as graph neural networks in the proposed multi-stage VAE approach and show that this method can reduce the Wasserstein distances between the property distributions of the test set and the generated set – and thus, by improving on manifold recovery, it can provide better results than its competitor methods.

[1]When $\sigma_\theta^2 = 1$, $-\log p_\theta(\mathbf{x}|\mathbf{z})$ can be simplified to $\|\mathbf{x} - \boldsymbol{\mu}_\theta\|_2^2$.



(a) Stage 1 **(b)** Stage 2 **(c)** Stage 3

Figure 4.2: Multi-stage VAE on synthetic data. The x-axis represents the norm of the data point and the y-axis represents the number of data points that are of x distance away from the unit sphere center. The histogram for an optimal manifold recovery should be a Dirac delta at location 1. The figure for stage 1 shows that most of the generated points fall *inside* of the sphere indicating poor manifold recovery. Training additional VAE stages improves manifold recovery as demonstrated by the concentration of points with norm of 1.

Synthetic Experiment We demonstrate that a multi-stage VAE setup improves the recovery of a ground-truth manifold with data generated according to Eq (4.6) from the surface of a 3-dimensional unit ball [Koehler et al., 2021]. The generated data are 3-dimensional unit vectors and form a 2-dimensional surface that we call *sphere* (because it has no volume). These vectors are then padded with 16 dimensions of zeros to embed the data on the low-dimensional manifold into an higher-dimensional ambient space:

$$x_i = [v, \vec{0}_{16}] \quad \text{where} \quad v' \sim \mathcal{N}(0, I_3) \text{ and } v = \frac{v'}{\|v'\|_2} = 1 \tag{4.6}$$

The intrinsic dimension of the data is 2 and the ambient dimension is 19. We trained on this data in three VAE stages and sample 1000 data points to visualize the results in the histograms. Theoretically, an optimal manifold recovery would only produce vectors of norm 1. Empirically, we observe that the VAE in the first stage *does not* recover the manifold, and many of the generated data points $\hat{x}_i$'s fall *inside* of the sphere, or $\|\hat{x}_i\|_2 < 1$, echoing the finding by Koehler et al. [2021]. In the second and third stage, however, we find that more data points fall *close to* the sphere (i.e. $\|\hat{x}_i\|_2 \approx 1$), indicating a better recovery of the manifold (Figure 4.2). The third-stage has slightly fewer points falling below the distance of 0.95 to the origin than the second in this experiment indicating a ceiling effect.

Application on Molecule Generation One of the major difficulties in generating molecules, is that while the latent space is distributed according to a Gaussian distribution, the observed data is discrete. In a simple discrete setting, the generated data follow a categorical distribution. Moreover, while the KL divergence term in the loss function is the same as for the above-discussed continuous VAE, the $\log p_\theta(\mathbf{x}|\mathbf{z})$ term in the discrete case is equivalent to the cross entropy loss. We index the categorical distribution's support with h:

$$\log p_\theta(\boldsymbol{x} = h|\mathbf{z}) = \log p_h = \sum_k t_k \log p_k = -H(\mathcal{T}, \mathcal{P}) \tag{4.7}$$

Algorithm 4 Training a Multi-Stage VAE

Given: moleculer dataset $\{\boldsymbol{x}_i | i = 1, 2, \cdots, n\}$
Initialize: A VAE Model $\mathcal{V}_1 = \{\phi^1, \theta^1\}$
Train $\mathcal{V}_1(\{\boldsymbol{x}_i | i = 1, 2, \cdots, n\})$
Generate the corresponding latent $\mathbf{z}_i^1 \sim q_{\phi^s}(\mathbf{z}^1 | \mathbf{x} = \boldsymbol{x}_i)$
Let s denote the stage number, $s = 2$ and $\sigma_\theta^1 = 0$
while $\sigma_\theta^s < 1$ **do**
    Initialize $\mathcal{V}_s = \{\phi^s, \theta^s\}$
    Train $\mathcal{V}_s(\{\mathbf{z}_i^{s-1} | i = 1, 2, \cdots, n\})$
    Generate latent $\mathbf{z}_i^s \sim q_{\phi^s}(\mathbf{z}^s | \mathbf{z}^{s-1} = \boldsymbol{z}_i^{s-1}) \ \ \forall i$
    $s = s + 1$
end while

where p_h is the output probability of the h-th element in the support and t_k is the k-th entry in the one-hot encoding of the true value. Though decoder variance is not a parameter in the log-likelihood function, it can still be measured based on the diversity of the output $\mathbf{x}$ given the same latent variable $\boldsymbol{z}$. We measured the diversity of outputs of three common molecule generation VAE methods and found that 0 for 2 of them have deterministic output, indicating that their decoder variance is 0 (Table 4.2).

Dai and Wipf [2019] observe two conditions in the first-stage VAE enabling improvements in later stage to occur: *i) the decoder variance converges to 0; and, ii) the entries in the diagonal of the encoder variance converge to either 0 or 1.* The first condition can be satisfied naturally despite the lack of variance parameter as discussed before. The continuous latent spaces of molecular VAE models allows us to verify second condition as well (Table 4.2). In Section 4.4, we provide empirical studies on the application of multi-stage VAE's in the molecule generation domain by evaluating the output quality using structural and property statistics [Polykovskiy et al., 2020]. We find that the multi-stage VAE generate molecules that are more similar in property to the test set. The precise steps to train and sample from a multi-stage VAE are described in Alg 4, 5 and as following:

In addition to Algorithm 4 and 5 in the main paper, we also include the following description of the training and sampling procedure.

1. Train a VAE on the molecular dataset $\{\boldsymbol{x}_i \,|\, i = 1, 2, \ldots n\}$, and upon convergence, save the latent variables $\mathbf{z}^1 \sim q_{\phi^1}(\mathbf{z}^1 | \mathbf{x} = \boldsymbol{x}_i)$ for all the molecules in the dataset;
2. With $\{\boldsymbol{z}_i^1 \,|\, i = 1, 2, \ldots n\}$ as input, train additional stages of VAE with tunable decoder variance and the latent variable at stage s is denoted as $\mathbf{z}^s$. We use feed-forward architectures for both the decoder $p_{\theta^s}(\mathbf{z}^s | \mathbf{z}^{s+1})$ and the encoder $q_{\phi^s}(\mathbf{z}^{s+1} | \mathbf{z}^s)$. They both follow Gaussian distributions. The dimension of $\mathbf{z}^s$ are the same across different stages. Repeat this step until the final stage, and the last-stage latent variable is denoted as $\mathbf{z}$.
3. During the sampling process, sample the latent representation of the last stage VAE, $\mathbf{z}^N \sim \mathcal{N}(0, I)$. Obtain the output from the stage-s decoder $\mathbf{z}^s \sim p_{\theta^s}(\mathbf{z}^s | \mathbf{z}^{s+1} = \boldsymbol{z}'^{s+1})$ as the input to the stage-$(s-1)$ decoder. Repeat until we reach the first stage VAE and get the new molecule sample $\boldsymbol{x}_i'$ from the first stage decoder via $\mathbf{x} \sim p_{\theta^1}(\mathbf{x} | \mathbf{z}^1 = \boldsymbol{z}'^1)$.

Algorithm 5 Sampling from a Multi-Stage VAE

Given: $\{V_s = \{\phi^s, \theta^s\} | s = 1, 2, \cdots, N\}$

Sampling $\mathbf{z}'^N_i \sim \mathcal{N}(0, 1) \ \ \forall i$

for s = N-1 to 1 **do**

  $\mathbf{z}'^s \sim p_{\theta^s}(\mathbf{z}^s | \mathbf{z}^{s+1} = \boldsymbol{z}'^{s+1}_i)$

end for

$\mathbf{x} \sim p_\theta(\mathbf{x} | \mathbf{z}^1 = \boldsymbol{z}'^1)$

return $\mathbf{x}$

We observe in our experiments that, for each additional stage of VAE we train, the decoder variance converges to a larger value than the previous one (details in Section 4.4.2). Also when the new stage's decoder variance converges to 1, the same as the prior $p(\mathbf{z})$'s variance, the improvements become minimal. In the next section, we experiment on three pre-existing VAE models for general molecule generation tasks. We verify if they meet the two conditions outlined for the continuous synthetic setting earlier. And the models that do are able to generate molecules more similar to the test set by training additional stages of VAE. We also show that the multi-stage VAE can increase the number of active molecules generated when fine-tuned for a protein target.

4.4 Experiments

In this section, we demonstrate the effectiveness of multi-stage VAE on two generation tasks. First, our algorithm learns to generate molecules by training on a large molecular database (i.e., ChEMBL), and we show that a multi-stage VAE generates molecules more similar to the test set in properties than a single stage VAE. Our baselines include a GNN model [Maziarz et al., 2021] that minimizes the losses between the true property values of the input molecules and the predicted property values of the output molecules for multiple targets during training and our method shows consistent improvements across various properties. Second, our algorithm is fine-tuned on two curated molecular datasets that are active against two different protein targets (JAK2 and EGFR). The multi-stage VAE improves the activity rate among the generated molecules. In this experiment, we compare our method against an RL-based fine-tuning method [Jin et al., 2020b] that uses activity predictor as the reward function and show that we can achieve equivalent or better results in much shorter time.

4.4.1 Unconstrained Generation

We adopt three existing VAE models as our first-stage VAE – hierarchical GNN [Jin et al., 2020a], MoLeR [Maziarz et al., 2021], and character-level RNN [Polykovskiy et al., 2020] – to compare the effects of multi-stage VAE on different model architectures featuring different molecule representations. We also adopt a GAN-based model [Prykhodko et al., 2019] as an additional comparison. We conduct experiments on the ChEMBL [Mendez et al., 2019] dataset consisting of 1,625k bioactive molecules with drug-like properties. It is split into training, testing, and validation datasets containing 1,463k, 81k, and 81k molecules respectively.

Model	Structural Statistics			Property Statistics			
	SNN ↑	Frag ↑	Scaf ↑	LogP ↓	SA ↓	QED ↓	MW ↓
HGNN	0.42	0.97	0.46	$0.92_{0.016}$	$0.070_{4.3e-3}$	$0.024_{9.5e-4}$	$68.8_{0.83}$
Multi-Stage HGNN	0.41	1.0	0.43	$\mathbf{0.095}_{0.019}$	$\mathbf{0.069}_{5.8e-3}$	$\mathbf{0.0067}_{1.0e-3}$	$\mathbf{5.0}_{0.72}$
MoLeR	0.41	0.96	0.48	$0.16_{6.78e-3}$	$\mathbf{0.028}_{1.96e-3}$	$0.047_{2.78e-3}$	$9.6_{0.70}$
Multi-Stage MoLeR	0.42	0.96	0.49	$\mathbf{0.087}_{6.16e-03}$	$0.031_{2.81e-03}$	$\mathbf{0.028}_{2.31e-03}$	$\mathbf{8.2}_{4.0e-01}$
RNN	0.38	1.0	0.38	$\mathbf{0.088}_{7.8e-3}$	$\mathbf{0.25}_{7.8e-3}$	$\mathbf{0.0088}_{1.6e-3}$	$3.2_{0.55}$
Multi-Stage RNN	0.38	1.0	0.36	$0.099_{5.5e-3}$	$0.27_{7.7e-3}$	$0.0099_{1.5e-3}$	$\mathbf{2.8}_{0.29}$
LatentGan	0.34	0.68	0.21	$0.69_{0.019}$	$0.63_{7.3e-3}$	$0.047_{2.0e-3}$	$27.2_{0.88}$
MoLeR + prop	0.43	0.97	0.49	$0.11_{1.03e-02}$	$0.13_{2.51e-03}$	$0.033_{8.65e-04}$	$6.6_{4.80e-01}$

Table 4.1: Properties of the generated molecules trained on the ChEMBL dataset.

The metrics used for the experiments in Section 4.4.1. Property Statistics include LogP (The Octanol-Water Partition Coefficient), SA (Synthetic Accessibility Score), QED (Quantitative Estimation of Drug-Likeness) and MW (Molecular Weight). These metrics determine the practicality of the generated molecules, for example, LogP measures the solubility of the molecules in water or an organic solvent [Wildman and Crippen, 1999], SA estimates how easily the molecules can be synthesized based on molecule structures [Ertl and Schuffenhauer, 2009], QED estimates how likely it can be a viable candidate of drugs [Bickerton et al., 2012]. The values listed in the table for each metric are the Wasserstein distances between the distributions of the property statistics in the test set and the generate molecule set.

Structural statistics include SNN (Similarity to Nearest Neighbor), Frag (Fragment Similarity), and Scaf (Scaffold Similarity). These statistics calculate two molecular datasets' structural similarity based on their extended-connectivity fingerprints [Rogers and Hahn, 2010], BRICS fragments [Degen et al., 2008] and Bemis–Murcko scaffolds [Bemis and Murcko, 1996].

The sample quality metrics are a lot more intuitive. Valid calculates the percentage of valid molecule outputs. Unique calculates the percentage of unique molecules in the first k molecules where $k = 1000$ for the ChEMBL dataset. Novelty calculates the percentage of molecules generated that are not present in the training set. FCD is the Fréchet ChemNet Distance [Preuer et al., 2018].

Below we provide the details of the models considered in this study:

Hierarchical GNN: This method extracts chemically valid motifs, or substructures, from the molecular graphs. The model consists of an encoder that encodes from atoms to motifs and a coarse-to-fine decoder that selects motifs to create the molecules. We use the configuration from the original model with a latent dimension of 20.

MoLeR GNN: Similarly to the hierarchical GNN, this method also extracts motifs in order to generate molecules piece by piece. The method's objective includes a term (Eq 4.8) that simultaneously learns a regression function $f_p(\cdot)$ to predict the ground-truth molecule properties y_p from the latent variables $\mathbf{z}$ during VAE training for each property p.

$$\mathcal{L}_{prop} = \sum_p \lambda_p \| f_p(\mathbf{z}) - y_p \|_2^2 \tag{4.8}$$

In our experiment we examine whether the inclusion of the loss term $\mathcal{L}_{prop}$ can improve the similarity of the generated molecules to the ground-truth data in properties (marked as "MoLeR +

prop" in Table 4.1). We remove $\mathcal{L}_{prop}$ to create a first-stage model "MoLeR" and implement our multi-stage method based on it. We use the original training configuration with a latent dimension of 64.

Vanilla RNN: The inputs to the model are SMILES strings and the vocabulary consists of the low-level symbols in the SMILES format. The encoder is a 1-layer GRU and the decoder is a 3-layer GRU. We use the original configuration with a latent dimension of 128.

Latent GAN: This is a 2-stage method. The first stage is a heteroencoder that takes SMILES strings as input while the second stage is a Wasserstein GAN with gradient penalty (WGAN-GP) that trains on the latent variables of the first stage encoder. We use the original parameters for training.

Results We sample 10,000 molecules from each model to generate the results in Table 4.1, which include structural and property statistics. Additional metrics such as validity are moved to Appendix 4.4.3 due to similar performance across all models. The entries in the property statistics section are the *Wasserstein distances* between the property distribution of the test set and the generated set. A lower value in these statistics signals increased similarity to the test set in these properties. All results are averaged over 6 sets of samples generated with different random seeds from the model. We include the standard deviations only for property statistics as all others exhibited small values (i.e., below 0.01).

The multi-stage HGNN improves upon HGNN by many folds on property statistics. The most notable improvements from the ChEMBL dataset are QED (from 0.024 to 0.0067), MW (from 68.8 to 5.0) and LogP (0.92 to 0.059). Structural statistics generally did not change substantially.

Multi-stage MoLeR improves upon standard MoLeR in three property metrics: LogP, QED and MW. Particularly in LogP and QED, the value goes down almost a half from 0.16 to 0.087 and from 0.047 to 0.028. The SA measure stays roughly the same, moving from 0.028 of MoLeR to 0.031 of multi-stage MoLeR. The regression term in the "MoLeR + prop" optimizes over LogP, SA and MW. For two of these three metrics, "MoLeR + prop" reaches lower statistics than MoLeR, where such regression term is left out of the objective, but the SA metric of "MoLeR + prop" is *more than four times higher* than without the regression term. This highlights the challenges of directly matching multiple statistics at once during training – the objectives could be conflicting with each other and lead to unexpected results. Eventually, the multi-stage MoLeR reaches lower property statistics than "MoLeR + prop" in three (LogP, SA and QED) out of four metrics. For the only property metric (MW) our algorithm performed worse (or bigger distance) in, the gap is only 24% worse than "MoLeR + prop" while for metric such as SA, our outputs are more than 300% better than "MoLeR+prop".

Multi-stage RNN performs worse than RNN in three out of four metrics. The RNN VAE's training process does not follow a standard VAE training procedure – the SMILES strings are input to both the encoder and decoder. This allows the decoder to rely less on the latent variables during the decoding process and the model exhibits signs of *posterior collapse* [Razavi et al., 2019a, Fu et al., 2019] during training. Our hypothesis for the poor performance of the multi-stage VAE with first-stage RNN model is that the variance of the first-stage decoder did not fulfill the condition of approaching 0 upon convergence (detailed analysis below).

Verification of Encoder and Decoder Variance Conditions We investigate how well each of the three first-stage models fulfills the two conditions (outlined in Section 4.3.2) hypothesized by Dai and Wipf [2019] as necessary for improvements in the later stages to occur. This examination can help explain their different behaviors. We present our findings in Table 4.2.

1. *The decoder variance of the first-stage model converges to zero*. Variance of a multinomial distribution is defined on each element in the support. To simplify the procedure, we substitute variance calculation with a simple experiment – input the same latent variable to the trained decoder 1,000 times, the number of distinct molecules the model generates can give us an indication on whether the decoder variance is approaching zero. We observe that for both of the GNN models, all 1,000 identical latent variables generate 1000 identical molecules (Table 4.2 column "Unique Outputs"). We can consider both models to have zero decoder variance. In contrast, the RNN model generates 1,000 distinctive SMILES strings. Thus, its decoder variance did not approach 0 and training multi-stage VAE did not improve the properties of generated molecules.
2. *Each entry of the encoder variance diagonal either converges to 0 or 1*. Considering a 0.1 tolerance interval around 0 and 1 and we find the following: The HGNN model's 20-dimensional encoder variance diagonal has all converged to 0; RNN's 128-dimensional encoder variance diagonal features 111 of 1's and 15 of 0's; and MoLeR's 64 dimensions of encoder variance diagonal has converged to 45 of 1's and 6 of 0's. The other of the dimensions fall somewhere in between 0.1 and 0.9 shown in Table 4.2's first three columns.

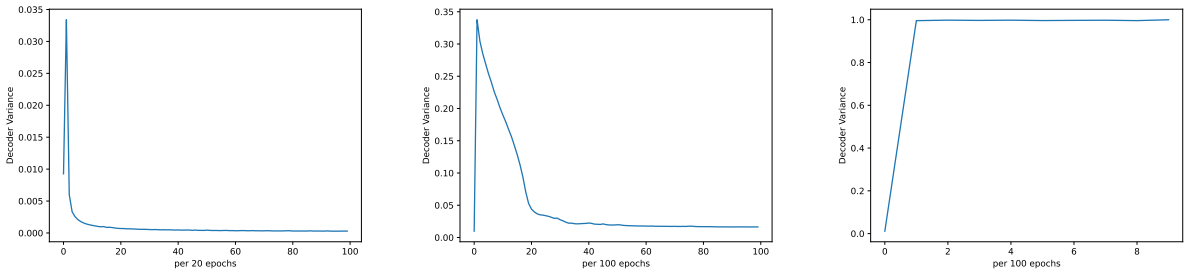
Model	Interval $x < 0.1$	Interval $0.1 \leq x \leq 0.9$	Interval $0.9 < x$	Unique Outputs
HGNN	20	0	0	1
MoLeR	6	13	45	1
RNN	15	2	111	1,000

Table 4.2: The first three columns examines whether the encoder variance Σ_ϕ's diagonal has converged to 0 or 1. The last column examines if the decoder variance σ_θ^2 converges to 0 based on the number of unique SMILES outputs after inputting 1,000 identical latent vectors.

We observe that the HGNN model meets both conditions for the simple continuous settings. As hypothesized, multi-stage HGNN improves significantly over HGNN on the second stage. While the MoLeR model fulfills the conditions only partially, multi-stage MoLeR still yields better results than single stage MoLeR. Its three stage variant achieves best performance. The RNN model does not meet either of the two conditions and training multi-stage RNN yield improvements. The detailed performances in each stage of the multi-stage VAE's is in Appendix 4.4.3. Overall, these findings corroborate the required conditions postulated by Dai and Wipf [2019].

4.4.2 Converged Decoder Variance in Different Stages

For HGNN model, the stage #2 model's decoder variance converges to 0.067 and the stage #3 decoder variance converges to 1.0. For the RNN-VAE model, the stage #2 model's decoder



(a) Stage 2 Decoder Variance **(b)** Stage 3 Decoder Variance **(c)** Stage 4 Decoder Variance

Figure 4.3: The decoder variance's change over the course of training time.

variance converges to 1.

For the MoLeR model, the stage #2 model's decoder variance converges to 0.00013, and the stage #3 model's decoder variance converges to 0.016. We train a stage #4 VAE and it converges to 1.0. We include the plots of the decoder variance during training at each stage in Figure 4.3. For both stage #2 and #3, the decoder variance briefly goes up in the beginning of the training before converging to a much smaller value. In stage #4, the decoder variance reaches 1 very quickly and the value stays unchanged. We include the results generated by a 4-stage VAE in Table 4.4's #4 row.

4.4.3 Additional Multi-stage VAE Results

In this section, we include additional results on multi-stage VAE. In Table 4.3, we included the full table of results on each stage of the multi-stage VAE as an extension to Table 4.1. It also includes the sample quality information. In addition, we also trained a multi-stage VAE on a polymer dataset [St. John et al., 2019] with HGNN as the first-stage. The first-stage result is included in the HGNN paper [Jin et al., 2020a]. We present the results from the second and third-stage VAE in relation to the first-stage in Table 4.5. We also trained two additional stages for MoLeR model with property matching included in the objective function in Table 4.6. An example output of 2-stage VAE on MNIST dataset is in Figure 4.4.

The Polymer Dataset[St. John et al., 2019] contains 86,353 polymers and it's divided into training, test and validation set that contains 76,353, 5000 and 5000 molecules each. Polymers generally have heavier weight than the molecules in the ChEMBL dataset and the dataset size is smaller. Uniqueness is selected to be at top $k = 500$ for the polymer dataset. On the polymer dataset, the second stage VAE improves significantly across all metrics – from 72.2 to 7.7 on MW, 0.020 to 0.0024 on QED, 0.089 to 0.031 on SA and 1.3 to 0.1 on LogP. In the third stage, 2 of the metrics (SA and QED) improved while the other 2 degraded.

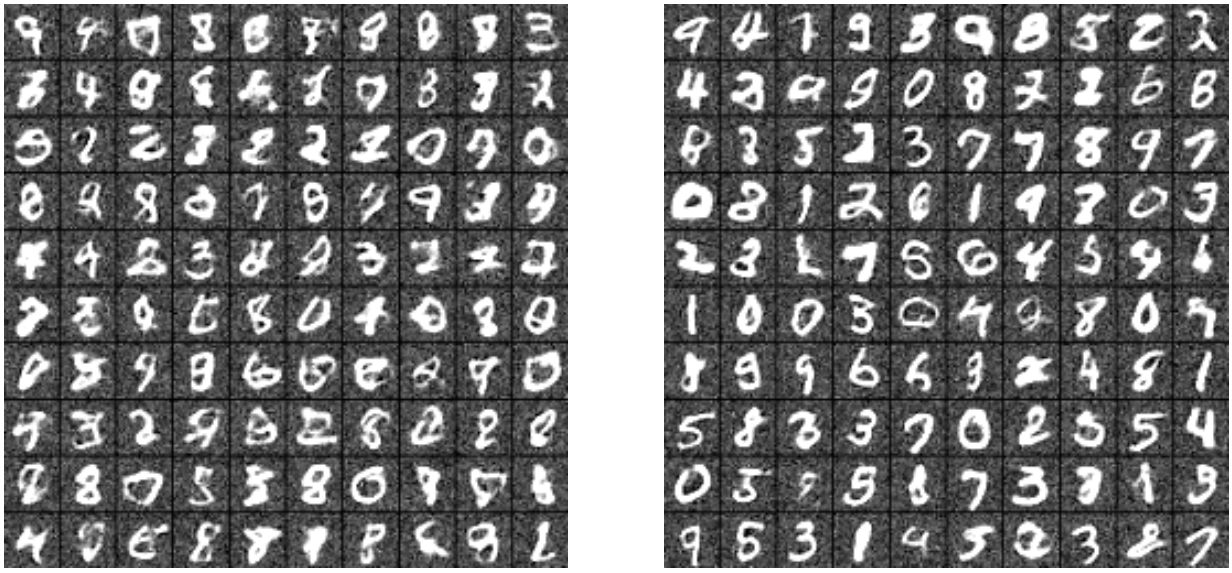
We train a multi-stage VAE on MoLeR with property matching as the first stage. Due to the modification to the objective function, none of the analysis described in the main paper necessarily apply here, but it is still interesting to see the results. In Table 4.6, we see that the second-stage VAE is able to improve upon the first-stage on all metrics while the third-stage improves only upon QED while the other 3 properties degraded.

Stage #	Sample Quality				Structural Statistics			Property Statistics			
	Valid ↑	Unique ↑	Novelty ↑	FCD ↓	SNN ↑	Frag ↑	Scaf ↑	LogP ↓	SA ↓	QED ↓	MW ↓
HGNN#1	1.0	1.0	0.99	5.1	0.42	0.97	0.46	$0.92_{0.016}$	$0.070_{4.3e-3}$	$0.024_{9.5e-4}$	$68.8_{0.83}$
HGNN#2	1.0	1.0	0.99	1.1	0.41	1.0	0.43	$0.095_{0.019}$	$\mathbf{0.069}_{5.8e-3}$	$\mathbf{0.0067}_{1.0e-3}$	$\mathbf{5.0}_{0.72}$
HGNN#3	1.0	1.0	1.0	1.2	0.41	1.0	0.46	$\mathbf{0.059}_{4.5e-3}$	$0.069_{6.3e-3}$	$0.016_{1.6e-3}$	$7.7_{0.42}$
MoLeR#1	1.0	1.0	0.99	2.1	0.41	0.96	0.48	$0.16_{6.78e-3}$	$\mathbf{0.028}_{1.96e-3}$	$0.047_{2.78e-3}$	$9.6_{0.70}$
MoLeR #2	1.0	1.0	0.99	2.2	0.42	0.96	0.53	$0.12_{6.13e-3}$	$0.041_{4.31e-3}$	$0.036_{9.25e-4}$	$6.8_{0.71}$
MoLeR #3	1.0	1.0	0.99	1.8	0.42	0.96	0.49	$\mathbf{0.087}_{6.16e-03}$	$0.031_{2.81e-03}$	$\mathbf{0.028}_{2.31e-03}$	$8.2_{4.0e-01}$
MoLeR + prop	1.0	1.0	0.99	2.1	0.43	0.97	0.49	$0.11_{1.03e-02}$	$0.13_{2.51e-03}$	$0.033_{8.65e-04}$	$\mathbf{6.6}_{4.80e-01}$
RNN#1	0.86	1.0	1.0	1.84	0.38	1.0	0.38	$\mathbf{0.088}_{7.8e-3}$	$\mathbf{0.25}_{7.8e-3}$	$\mathbf{0.0088}_{1.6e-3}$	$3.2_{0.55}$
RNN#2	0.87	1.0	1.0	1.86	0.38	1.0	0.36	$0.099_{5.5e-3}$	$0.27_{7.7e-3}$	$0.0099_{1.5e-3}$	$\mathbf{2.8}_{0.29}$
LatentGan	0.77	0.98	0.99	17.3	0.34	0.68	0.21	$0.69_{0.019}$	$0.63_{7.3e-3}$	$0.047_{2.0e-3}$	$27.2_{0.88}$

Table 4.3: Properties of the generated molecules trained on the ChEMBL dataset.

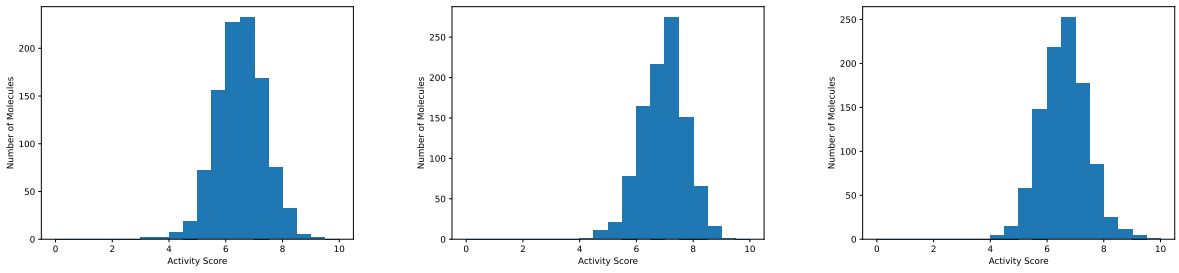
MoLeR + prop	Sample Quality				Structural Statistics			Property Statistics			
	Valid ↑	Unique ↑	Novelty ↑	FCD ↓	SNN ↑	Frag ↑	Scaf ↑	LogP ↓	SA ↓	QED ↓	MW ↓
#1	1.0	1.0	0.99	2.1	0.41	0.96	0.48	$0.16_{6.78e-3}$	$\mathbf{0.028}_{1.96e-3}$	$0.047_{2.78e-3}$	$9.6_{0.70}$
#2	1.0	1.0	0.99	2.2	0.42	0.96	0.53	$0.12_{6.13e-3}$	$0.041_{4.31e-3}$	$0.036_{9.25e-4}$	$6.8_{0.71}$
#3	1.0	1.0	0.99	1.8	0.42	0.96	0.49	$0.087_{6.16e-03}$	$0.031_{2.81e-03}$	$\mathbf{0.028}_{2.31e-03}$	$8.2_{4.0e-01}$
#4	1.0	1.0	0.99	1.9	0.42	0.96	0.48	$\mathbf{0.066}_{7.39e-03}$	$0.030_{3.27e-03}$	$0.029_{1.52e-03}$	$10._{6.44e-01}$

Table 4.4: Properties of the generated molecules from the 4th-stage VAE trained on the ChEMBL dataset using MoLeR without property matching as the first stage.

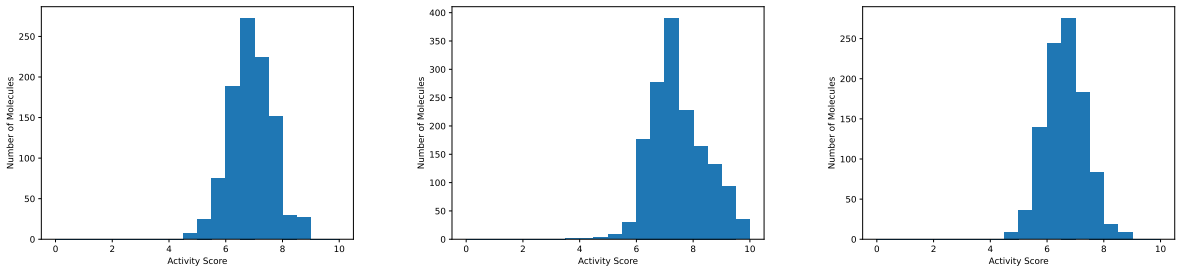


(a) Stage 1 **(b)** Stage 2

Figure 4.4: Output of 1- and 2-stage VAE on MNIST data.

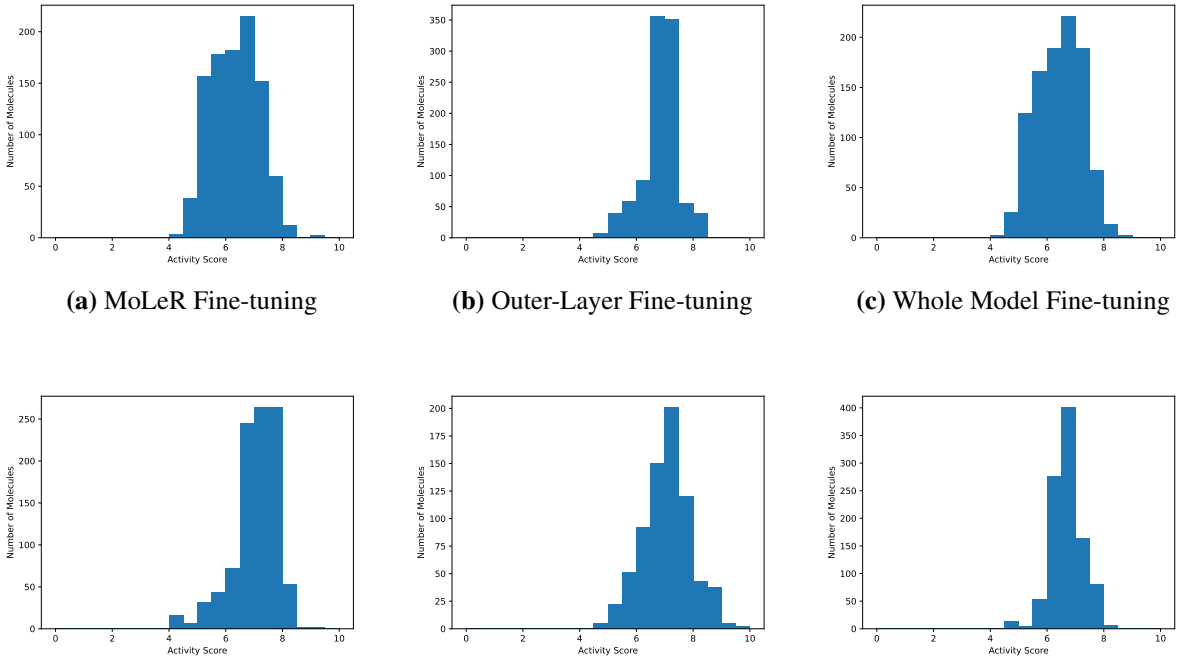


(a) MoLeR Fine-tuning **(b)** Outer-Layer Fine-tuning **(c)** Whole Model Fine-tuning



(d) Inner-Layer Fine-tuning **(e)** Test Set **(f)** RL Fine-tuning.

Figure 4.5: The distributions of the generated molecules' activity scores by the Chemprop model on the JAK2 protein in six histograms.



(a) MoLeR Fine-tuning **(b)** Outer-Layer Fine-tuning **(c)** Whole Model Fine-tuning

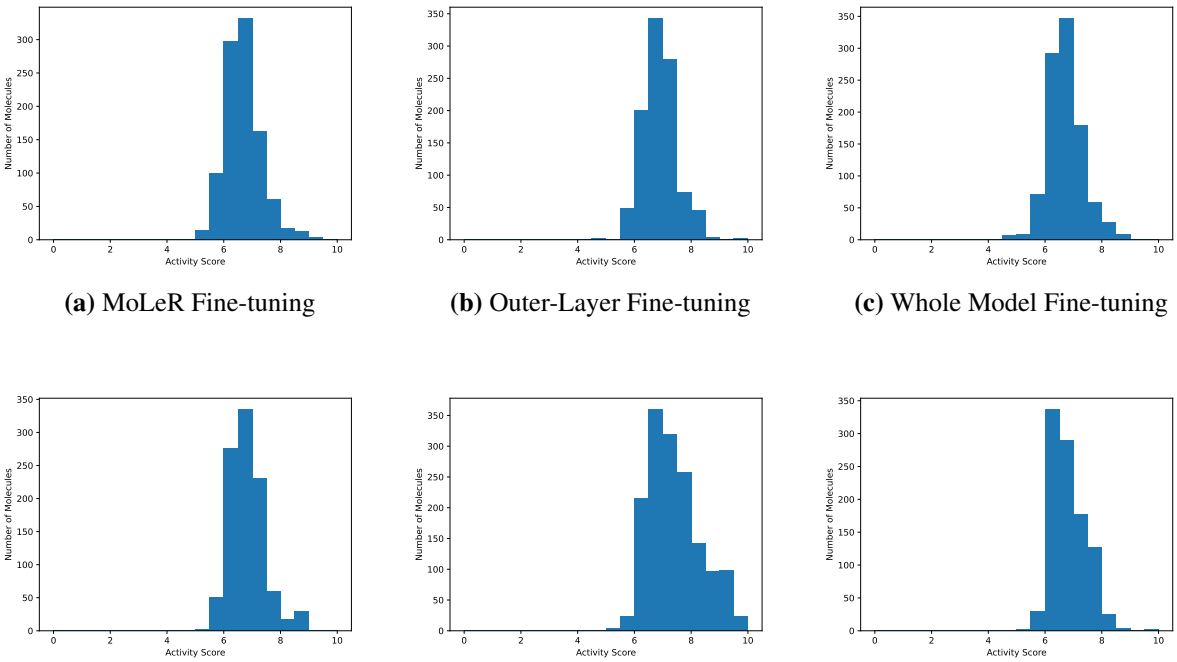
(d) Inner-Layer Fine-tuning **(e)** Test Set **(f)** RL Fine-tuning.

Figure 4.6: The distributions of the generated molecules' activity scores by the Random Forest model on the EGFR protein in six histograms.

4.4.4 Generation for a Protein Target

In addition to the unconstrained generation of molecules, we explore generating molecules for specific protein targets. We pre-train a MoLeR model and a multi-stage MoLeR on the ChEMBL dataset and fine-tune them on two curated, and much smaller, datasets [Korshunova et al., 2022] consisting of molecules that are active inhibitors of Janus Kinas 2 (JAK2) and inhibitors of the Epidermal Growth Factor (EGFR). For each of the protein targets, the regression dataset contains the molecules and their corresponding activity scores (from 0 to 10). A score above 6 is considered active. The dataset size is around 19k for JAK2, with about 15.6k active molecules, and around 15k for EGFR, with about 7.8k active molecules. Both active and non-active molecules are used to train a regressor of the activity score while only the active ones are used to fine-tune the VAE. We divide each dataset into 8:1:1 for training, validation, and testing for both VAE and predictor. In addition, there is a separate classification dataset for each protein target – 60k for JAK2 and 50k for EGFR – containing only the binary activity information of each molecule. The dataset is divided into 9:1 for training and testing of the classifiers.

Metrics We evaluate our method and baseline methods in three major categories: activity, diversity, and novelty. All evaluations are reported with means and standard deviations across 5 datasets with 1,000 molecules each generated by different random seeds. Particularly for activity scores, we report both the mean score and percentage of active molecules (6 as the cutoff point) in a dataset by a Chemprop regressor [Yang et al., 2019], and the percentage of active molecules



(a) MoLeR Fine-tuning **(b)** Outer-Layer Fine-tuning **(c)** Whole Model Fine-tuning

(d) Inner-Layer Fine-tuning **(e)** Test Set **(f)** RL Fine-tuning.

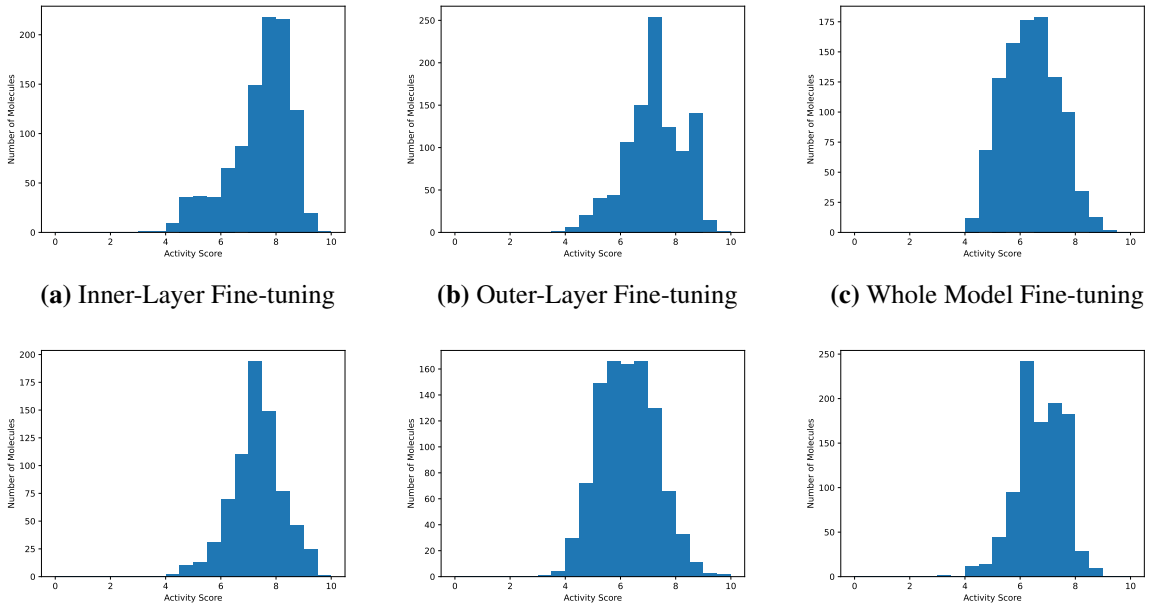
Figure 4.7: The distributions of the generated molecules' activity scores by the Random Forest model on the JAK2 protein in six histograms.

by a Chemprop classifier. The regressor reaches an RMSE of 0.5 for the JAK2 dataset and 0.6 for the EGFR dataset. As reference, performance metrics for all predictors on the active test set split are provided by Tables 4.7a and 4.7b. Novelty is defined as the fraction of molecules with nearest neighbor similarity in the active training set below 0.4. Diversity is calculated based on the pairwise molecular distance $\text{sim}(X, Y)$ within the generated dataset. The function $\text{sim}(\cdot, \cdot)$ is defined as the Tanimoto distance over Morgan fingerprints of two molecules. These two metrics are defined as follows [Jin et al., 2020b]:

$$\text{Diversity} = 1 - \frac{2}{n(n-1)} \sum_{X,Y} \text{sim}(X, Y) \tag{4.9}$$

$$\text{Novelty} = \frac{1}{n} \sum_{\mathcal{G}} \mathbf{1}[\text{sim}(\mathcal{G}, \mathcal{G}_{\mathcal{SNN}}) < \mathbf{0.4}] \tag{4.10}$$

Methods We compare the generated molecules from the fine-tuned multi-stage MoLeR model to the fine-tuned one-stage MoLeR model. We include RationaleRL [Jin et al., 2020b] as a reinforcement learning-based baseline. We obtained the multi-stage MoLeR from training on the full ChEMBL dataset as described in Section 4.3.2. The MoLeR model was fine-tuned on the curated active molecules dataset after pretraining on the full ChEMBL dataset. Each stage of the multi-stage MoLeR is initialized with the parameters from the corresponding stage of the pre-trained model and fine-tuned on the encoded latent variables of the curated dataset from the



(a) Inner-Layer Fine-tuning **(b)** Outer-Layer Fine-tuning **(c)** Whole Model Fine-tuning

(d) Test set **(e)** MoLeR Fine-tuning **(f)** RationaleRL

Figure 4.8: The distributions of the generated molecules' activity scores on the EGFR protein predicted by the Chemprop model. We include 5 sets of molecules generated by different methods as well as the ground-truth test set. The x-axis represents the activity score ranging from 0 to 10 and the y-axis is the number of molecules in each bin.

fine-tuned previous-stage VAE. We fine-tuned the later-stage VAEs in three ways: fine-tuning the entire model, fine-tuning only the two inner layers connecting to the latent sampling layer, and fine-tuning only the two outer layers connecting to the input and output. We visually compare the distributions of the activity scores for EGFR from the three types of fine-tuning methods and baselines in Figure 4.8. The quantitative metrics are shown in Table 4.7a and 4.7b. Each evaluation reports metrics for the active test set as a reference.

Results For both of the protein targets, fine-tuning the multi-stage MoLeR in either of the three ways produces *more* active molecules and *higher* mean activity scores of the protein targets than fine-tuning only the first-stage VAE (Table 4.7a and 4.7b) by both the classifier and the regressor. The improvement is especially pronounced on the EGFR dataset, which is about half of the JAK2 dataset's size. Particularly, fine-tuning only the inner layers or outer layers produces more active molecules than fine-tuning the whole model. Their activity score distributions are also more similar to the test set's (Figure 4.8). The peak of the activity score distribution around 7 is well-captured by the two fine-tuning methods of the VAE in Figure 4.8a and 4.8b. Fine-tuning only two layers reduces the number of training parameters with the same amount of data, thus leading to better results.

The RationaleRL model is initialized with a generative model trained on the ChEMBL dataset and then fine-tuned with an RL algorithm. A Random-Forest classifier with Morgan fingerprint features is used as its reward signal Jin et al. [2020b]. Evaluations by the random forest model are

in Section 4.5. Our methods have higher mean activity scores than RationaleRL on both datasets. On the EGFR dataset, the fine-tuned multi-stage MoLeR reaches a higher level of activity than RationaleRL by both the regressor and the classifier. On the JAK2 dataset, the regressor has predicted higher activity levels on the generated molecules from our methods than RationaleRL but our methods have lower activity levels as predicted by the classifier. This discrepancy could be attributed to a bias towards negative samples due to an imbalanced training size and poor out-of-distribution generalization as evidenced by the high novelty scores of our methods. Fine-tuned multi-stage MoLeRs are higher on novelty metric than RationaleRL with similar diversity levels. In addition, our method is much more computationally efficient to fine-tune, taking only a few hours as opposed to days for RationaleRL.

4.5 Experiment Results from Random Forest

The RationaleRL model is initialized with a generative model trained on the ChEMBL dataset and then fine-tuned with an RL algorithm. A Random-Forest classifier with Morgan fingerprint features is used as its reward signal [Jin et al., 2020b] in the fine-tuning process. We include the prediction results by the Random Forest regressor (RFR) and classifier (RFC) in Table 4.8 and Table 4.9 as an extension to Table 4.7a and Table 4.7b. RationaleRL generally has a higher percentage of active molecules when predicted by the two Random Forest models than Chemprop, a sign of possible overfitting. No such pattern is observed for our methods which did not involve property predictors in the fine-tuning process. On the EGFR dataset, RationaleRL reaches a higher level of activity than fine-tuned multi-stage VAE as predicted by Random Forest models but a lower level of activity when predicted by Chemprop. On the JAK2 dataset, our methods have lower activity levels than RationaleRL as predicted by both classifiers but both regressors have predicted similar or slightly higher activity levels on the generated molecules from our methods than RationaleRL. Our method reaches slightly higher mean activity scores than RationaleRL on both datasets predicted by both regressors.

4.6 Training details on multi-stage VAE

Each stage of the multi-stage VAE with HGNN as the first stage has three fully-connected layers of size 512 for both encoders and decoders in addition to the input and output layer which are of size 20 (latent dimensions). The initial decoder variance is set at 0.05. Learning rate is set at 0.0001.

Each stage of the multi-stage VAE with MoLeR as the first stage has five fully-connected layers of size 1025 for both encoders and decoders in addition to the input and output layer which are of size 64 (latent dimensions). The initial decoder variance is set at 0.007. Learning rate is set at 0.0001. We trained our model for 10000 epochs but fewer epochs (e.g. 5000) can probably achieve similar results. For fine-tuning, the decoder variance is held as constant during the process. Fine-tuning either inner-layer or extra-layer is done by loading the pre-trained model and add two extra layers either connecting to the latent layer or the output. The two extra layers are randomly

[1]Training on a single "NVIDIA GeForce RTX 2080 Ti" GPU.

initialized. The pre-trained part of the model is frozen while only the additional layers are being trained. Fine-tuning the whole model means to load the pre-trained model and only freeze the decoder variance while training the rest of it without additional layers. During fine-tuning, we use 0.00001 as the learning rate and train for 50000 epochs.

4.7 Conclusion

In this chapter, we tackle the challenging problem of improving one or multiple molecular properties from pre-existing VAE methods by presenting a multi-stage VAE model shown to improve manifold recovery in a synthetic experiment. We demonstrate our contributions on i) an unconstrained generation experiment on ChEMBL dataset in which the multi-stage VAEs are able to improve upon their conventional stage-1 counterparts on property statistics such as LogP or SA; and ii) a finetuning experiment that shows how refined multi-stage VAE's yield more active molecules for protein targets than their refined stage-1 counterparts. Our model achieves improvements on various molecular properties while in training being agnostic to any particular property objectives, meaning, no property specific optimization objectives are required during training. Yet, it is able to achieve comparable, and sometimes better outcomes than specialized approaches that directly optimize over one or multiple property objectives with the help of property predictors.

	Sample Quality				Structural Statistics			Property Statistics			
HGNN	Valid ↑	Unique ↑	Novelty ↑	FCD ↓	SNN ↑	Frag ↑	Scaf ↑	LogP ↓	SA ↓	QED ↓	MW ↓
#1	1.0	1.0	0.57	0.62	0.67	0.98	0.37	$1.3_{0.030}$	$0.089_{3.0e-3}$	$0.020_{1.2e-3}$	$72.2_{1.42}$
#2	1.0	1.0	0.51	0.27	0.69	0.99	0.37	$\mathbf{0.10}_{0.033}$	$0.031_{3.3e-3}$	$0.0041_{9.5e-4}$	$\mathbf{7.7}_{1.1}$
#3	1.0	1.0	0.52	0.29	0.69	0.99	0.38	$0.24_{0.017}$	$\mathbf{0.024}_{4.1e-3}$	$\mathbf{0.0024}_{2.9e-4}$	$9.4_{2.3}$

Table 4.5: Properties of the generated molecules trained on the polymers dataset.

MoLeR	Sample Quality				Structural Statistics			Property Statistics			
+ prop	Valid ↑	Unique ↑	Novelty ↑	FCD ↓	SNN ↑	Frag ↑	Scaf ↑	LogP ↓	SA ↓	QED ↓	MW ↓
#1	1.0	1.0	0.99	2.1	0.43	0.97	0.49	$0.11_{1.03e-02}$	$0.13_{2.51e-03}$	$0.033_{8.65e-04}$	$6.6_{4.80e-01}$
#2	1.0	1.0	0.99	2.2	0.42	0.97	0.48	$\mathbf{0.080}_{1.37-02}$	$\mathbf{0.090}_{6.03-03}$	$0.030_{1.80-03}$	$\mathbf{6.0}_{2.94-01}$
#3	1.0	1.0	0.99	2.0	0.43	0.97	0.41	$0.096_{1.21-02}$	$0.13_{5.17-03}$	$\mathbf{0.022}_{1.68-03}$	$9.1_{7.34-01}$

Table 4.6: Properties of the generated molecules from the muli-stage VAE trained on the ChEMBL dataset using MoLeR with property matching as the first stage.

Model Type	Classifier Activity ↑	Regressor Activity ↑	Mean Score ↑	Diversity ↑	Novelty ↑	Time ↓
Active Test set	0.93	0.92	7.4	0.85	0.016	-
MoLeR	$0.44_{1.18\,e-02}$	$0.57_{1.21\,e-02}$	$6.3_{2.94\,e-02}$	$\mathbf{0.87}_{2.26\,e-03}$	$\mathbf{0.63}_{1.81\,e-02}$	4 min
RationaleRL	$0.79_{7.94\,e-03}$	$0.84_{1.45\,e-02}$	$6.8_{1.56\,e-02}$	$0.79_{1.38\,e-03}$	$0.055_{5.24\,e-03}$	4.5 d
Fine-tuned Multi-Stage Models						
Whole-Model	$0.53_{6.43\,e-03}$	$0.63_{6.65\,e-03}$	$6.4_{3.18\,e-02}$	$0.85_{9.95\,e-04}$	$0.54_{9.65\,e-03}$	1.5 h
Inner-Layer	$0.79_{1.61\,e-02}$	$0.87_{1.31\,e-02}$	$\mathbf{7.4}_{3.92\,e-02}$	$0.71_{4.41\,e-03}$	$0.17_{2.03\,e-02}$	1.25 h
Outer-Layer	$\mathbf{0.85}_{1.30\,e-02}$	$\mathbf{0.89}_{5.31\,e-03}$	$7.2_{3.17\,e-02}$	$0.71_{5.30\,e-03}$	$0.22_{8.78\,e-03}$	1.5 h

(a) EGFR

Model Type	Classifier Activity ↑	Regressor Activity ↑	Mean Score ↑	Diversity ↑	Novelty ↑	Time ↓
Active Test set	0.97	0.97	7.5	0.88	0.016	-
MoLeR	$0.52_{1.15e-02}$	$0.74_{6.89e-03}$	$6.5_{1.17e-02}$	$\mathbf{0.90}_{5.62e-04}$	$\mathbf{0.85}_{6.71e-03}$	7 min
RationaleRL	$\mathbf{0.85}_{8.96e-03}$	$0.79_{1.67e-02}$	$6.6_{2.93e-02}$	$0.87_{2.58e-03}$	$0.24_{7.42e-03}$	8.5 d
Fine-tuned Multi-Stage Models						
Whole-Model	$0.54_{8.31e-03}$	$0.78_{1.02e-02}$	$6.6_{2.22e-02}$	$0.89_{4.47e-04}$	$0.82_{7.80e-03}$	2.8 h
Inner-Layer	$0.66_{7.58e-03}$	$0.88_{1.11e-02}$	$\mathbf{6.9}_{1.01e-02}$	$0.88_{8.39e-04}$	$0.78_{6.29e-03}$	2.8 h
Outer-Layer	$0.62_{2.27e-02}$	$\mathbf{0.89}_{8.37e-03}$	$\mathbf{6.9}_{3.72e-02}$	$0.86_{1.29e-03}$	$0.69_{5.37e-03}$	2.8 h

(b) JAK2

Table 4.7: Evaluation of the generated molecules targeting EGFR and JAK2 by three multi-stage MoLeR fine-tuning methods: fine-tuning the whole model, fine-tuning only inner layers, and fine-tuning outer layers. They are compared against baseline models such as fine-tuned MoLeR and RationaleRL. The evaluation metrics include the percentage of active molecules, mean activity scores, diversity, and novelty.

Model Type	Activity (RFC)	Activity (RFR)	Mean (RFR)
Active Test set	0.98	0.89	7.1
MoLeR	$0.41_{1.14\,e-02}$	$0.60_{1.46\,e-02}$	$6.3_{2.16\,e-02}$
RationaleRL	$0.88_{1.23\,e-02}$	$0.94_{4.50\,e-03}$	$6.7_{1.42\,e-02}$
Whole-Model	$0.50_{1.38\,e-02}$	$0.69_{4.31\,e-03}$	$6.5_{1.91\,e-02}$
Inner-Layer	$0.88_{1.27\,e-02}$	$0.90_{8.40\,e-03}$	$7.0_{2.04\,e-02}$
Outer-Layer	$0.86_{1.75\,e-02}$	$0.89_{1.17\,e-02}$	$6.9_{2.61\,e-02}$

Table 4.8: Evaluation of the generated molecules targeting EGFR by the random forest method with three multi-stage VAE fine-tuning methods: fine-tuning the whole model, fine-tuning only the inner-layers and fine-tuning the outer-layers. They are compared against baseline models such as fine-tuned one-stage MoLeR and RationaleRL.

Model Type	Activity (RFC)	Activity (RFR)	Mean (RFR)
Active Test set	0.99	0.98	7.4
MoLeR	$0.44_{1.73e-02}$	$0.89_{1.24e-02}$	$6.7_{2.44e-02}$
RationaleRL	$0.92_{4.93e-03}$	$0.96_{7.26e-03}$	$6.8_{1.67e-02}$
Whole-Model	$0.46_{1.13e-02}$	$0.91_{5.90e-03}$	$6.7_{1.16e-02}$
Inner-Layer	$0.56_{6.68e-03}$	$0.94_{2.28e-03}$	$6.8_{1.36e-02}$
Outer-Layer	$0.65_{1.23e-02}$	$0.95_{1.96e-03}$	$6.9_{1.53e-02}$

Table 4.9: Evaluation of the generated molecules targeting JAK2 by the random forest method with three multi-stage VAE fine-tuning methods: fine-tuning the whole model, fine-tuning only the inner-layers and fine-tuning the outer-layers. They are compared against baseline models such as fine-tuned one-stage MoLeR and RationaleRL.

Chapter 5

Non-Differentiable Diffusion Guidance for Improved Molecular Geometry

5.1 Introduction

Applications of generative models in feature-rich geometries have the potential to accelerate scientific discoveries in chemistry, biology, and materials science. For example, the *in silico* generation of 3D geometries for molecules and proteins can help screen novel drug candidates and model the protein-molecule interaction to accelerate drug discovery Corso et al. [2022], Jumper et al. [2021], Xu et al. [2023], Hoogeboom et al. [2022].

In particular, molecules can be modeled with a graph, each node of the graph representing an atom and containing feature information such as 3D coordinates and atom types. The geometries of the generation results have domain-specific implications – a molecule's stability and properties depend significantly on its preferred quantum geometric states, i.e., atomic and molecular geometries. For example, polarity is closely related to molecular geometry. A water molecule H_2O has a stable V-shaped H-O-H geometry of 104.5 degrees and is thus polar. A generated H_2O molecule with a linear H-O-H geometry would instead be nonpolar but unstable (as demonstrated in Figure 5.1). Therefore, when we discuss molecules and their properties, it is essential to start from their preferred stable geometries. In this chapter, we propose to improve generated molecular geometries by incorporating a non-differentiable black-box predictor in the diffusion sampling process.

Many diffusion model approaches have been applied to 3D molecular structures [Hoogeboom et al., 2022, Xu et al., 2023, Bao et al., 2022, Vignac et al., 2022]. The generation results are evaluated on general stability and validity as well as on the performance of property-conditioned generation, all of which depend on the geometry of the resulting molecules. To better control the generation results, Hoogeboom et al. [2022], Xu et al. [2023] have proposed to train conditional generative models where they input the property values as conditions into the model during training and sampling to obtain novel molecules fulfilling the requirement. [Han et al., 2023] applied the training-free regressor-guided diffusion sampling method, which has been shown to improve image conditional generation quality [Bansal et al., 2023, Dhariwal and Nichol, 2021], to the task of molecule conditional generation and has seen remarkable improvements. The guidance

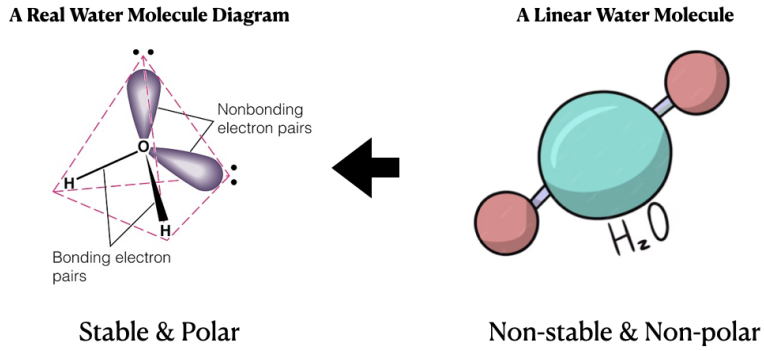


Figure 5.1: Left: A water molecule in real life that forms a 104.5 degree angle between the two HO bond. The molecule is polar and stable. Right: A linear water molecule that is nonpolar and nonstable. Our goal is to optimize the molecular geometry during the sampling process as optimising the molecular geometry after generation could change the properties of the generated molecules.

method uses the gradient of classifier or regressor loss between the conditioned value and the generated result's predicted value to perform "correction" during the sampling process to guide the trajectory toward a result that fulfills the conditioned.

We propose to generalize the use of a neural predictor for guiding the diffusion model generation to a *non-differentiable* expert oracle that we can query from. To improve the generated molecule geometries, we use an external quantum chemistry package *xTB* with the GFN2-xTB method [Bannwarth et al., 2019] that conducts accurate and efficient quantum chemistry calculation of atom forces and we use a two-point method [Nesterov and Spokoiny, 2017] to estimate the gradient of the data point in the chemical property value landscape. By guiding the sampling process towards an output that minimizes forces on each atom, we naturally achieve geometry refinement as an end result. Directly incorporating an expert oracle removes the estimation error of a predictor. Even in the unlikely event that the predictor achieves perfect accuracy on the test set, it is not guaranteed to achieve good accuracy on an unseen dataset with a different distribution. For example, the molecules from QM9 [Ramakrishnan et al., 2014] are already stable molecules with optimal energies and minimized force on each atom, which is generally not the truth for the molecules generated during the denoising process of a diffusion model. In this chapter, we present our non-differentiable diffusion guidance method and apply it to the problem of molecule generation for the purpose of geometry optimization. To evaluate the effectiveness of our proposed approach, we show that our non-differentiable oracle guidance can improve validity and stability among the generated molecules. When used alongside other neural property regressors for multi-objective conditional generation, our method yields molecules with more on-point property values, while maintaining small net forces and high validity.

5.2 Related Work

This chapter lies in the intersection of predictor-guided diffusion generation and molecule generation for drug discovery. Various types of generative models have been proposed to model molecular data, including VAE [Jin et al., 2018, 2019, Kusner et al., 2017, Maziarz et al., 2021]

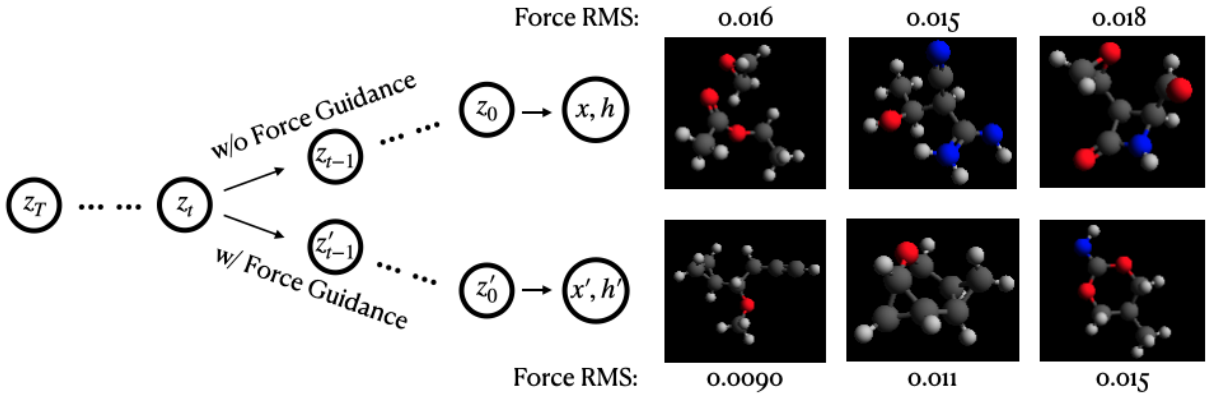


Figure 5.2: Visualization of molecule generation with and without force regularization. Guidance is estimated with the non-differentiable chemistry oracle xTB package. We only started to add guidance from timestep t to 0 and t is selected as 400 based on experiment results as well as prior literature [Han et al., 2024]. With force guidance, we are able to generate molecules with smaller net force on each atom and are

and GAN [Prykhodko et al., 2019, De Cao and Kipf, 2018]. However, these models focus on generating molecules as 2D graphs without 3D coordinate information. The autoregressive model is one approach to learning to generate 3D molecules including G-Schnet [Gebauer et al., 2019] and G-SphereNet [Luo and Ji, 2022], however, they are less effective and powerful compared to diffusion models [Hoogeboom et al., 2022]. The equivariant diffusion model EDM for 3D molecule generation was first proposed by Hoogeboom et al. [2022], which utilizes an equivariant graph neural network to model the molecules as graphs with coordinates and atom types as node features. GeoLDM [Xu et al., 2023] further extends EDM to a latent diffusion architecture and has shown improvements in stability and validity. Han et al. [2023] proposed a guided diffusion model but they assume dependency on different properties, which is usually hard to define ahead of guidance, and unlike our method, their method requires differentiable neural models to provide guidance, which can not deal with non-differentiable oracles and needs extra training to achieve good guidance. However, to achieve conditional generation, both EDM and GeoLDM need to be re-trained, where the target property value is appended to the feature space to generate molecules that fulfill certain property requirements.

Guided diffusion generation has shown promising results in the domain of image [Dhariwal and Nichol, 2021, Bansal et al., 2023, Zhang et al., 2023, Rombach et al., 2022], where its generation can be conditioned on texts [Rombach et al., 2022], poses and edges [Zhang et al., 2023], and classifiers [Dhariwal and Nichol, 2021]. A similar approach is adopted for property-guided molecule generation [Vignac et al., 2022, Bao et al., 2022, Han et al., 2023]. Without re-training a conditional model from scratch, Vignac et al. [2022], Han et al. [2023] trained additional differentiable neural regressors of properties, and use the gradients as guidance during the sampling process of an unconditional diffusion model. Bao et al. [2022] proposed to train time-dependent regressors of properties as guidance during sampling. This chapter differs from the previous work by introducing an approach to directly estimate the guidance signal (e.g., the gradient) from a non-differentiable expert oracle without training additional neural predictors.

5.3 Preliminary

In this section, we will introduce the diffusion model, explain how to achieve equivariance among the generated molecules using diffusion models, and discuss the architecture we use. We will also discuss the semi-empirical quantum mechanic method GFN2-xTB used as guidance in this chapter.

Diffusion Models In general, a diffusion model [Ho et al., 2020, Song et al., 2020a, Dhariwal and Nichol, 2021, Sohl-Dickstein et al., 2015] consists of a forward *diffusion process* and a reverse *denoising process*. The diffusion process is a Markov chain that gradually adds Gaussian noises with a variance schedule $\beta_{1:T}$ from timestep 1 to T to the original datapoint $\boldsymbol{x}_0$. The schedule is chosen such that $\boldsymbol{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The forward diffusion process q is usually defined as a fixed schedule by the following:

$$q(\boldsymbol{x}_{1:T} \mid \boldsymbol{x}_0) = \prod_{t=1}^{T} q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}) \qquad q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \mathbf{I}) \tag{5.1}$$

where $\beta_{1:T}$ is pre-defined ahead of training. The reverse denoising process starts with $\boldsymbol{x}_T$ and recovers the original datapoint $\boldsymbol{x}_0$ by predicting the mean of $\boldsymbol{x}_{t-1}$ given $\boldsymbol{x}_t$, denoted as $\mu_\theta(\boldsymbol{x}_t, t)$ where θ is the parameter. We can model the reverse process as:

$$p_\theta(\boldsymbol{x}_{0:T}) = p(\boldsymbol{x}_T) \prod_{t=1}^{T} p_\theta(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t) \qquad p_\theta(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{x}_{t-1}; \mu_\theta(\boldsymbol{x}_t, t), \Sigma_\theta(\boldsymbol{x}_t, t)) \tag{5.2}$$

In practice, Σ_θ is set to be $\sigma_t^2 I$ for all t for simplicy, and $\sigma_t = \sqrt{1-\alpha_t^2}$ and $\alpha_t = \sqrt{\prod_{i=1}^{t}(1-\beta_i)}$. Ho et al. [2020] further simplified the objective from predicting mean $\mu_\theta(\boldsymbol{x}_t, t)$ to predict the noise at each step, which is denoted by $\epsilon_\theta(\boldsymbol{x}_t, t)$, as $\boldsymbol{x}_t = \alpha_t \boldsymbol{x}_0 + \sigma_t \epsilon$, and $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The training objective is minimize $\mathbb{E}_{\boldsymbol{x}_0, \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), t}\left[\|\epsilon - \epsilon_\theta(\boldsymbol{x}_t, t)\|^2\right]$, and the original $\mu_\theta(\boldsymbol{x}_t, t)$ can be parameterized as $\frac{1}{1-\beta_t}(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t^2}}\epsilon_\theta(\boldsymbol{x}_t, t))$. Consequently, we have

$$\boldsymbol{x}_{t-1} \sim \mathcal{N}(\frac{1}{1-\beta_t}(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t^2}}\epsilon_\theta(\boldsymbol{x}_t, t)), \rho_t^2 \mathbf{I}) \tag{5.3}$$

We follow the implementation by Xu et al. [2023] where $\rho_t = \sqrt{\frac{\sigma_{t-1}}{\sigma_t}}\beta_t$ is a predefined variance.

Latent Diffusion Architecture for 3D Molecule Generation An N-atom molecule can be represented as a point cloud $\mathcal{G} = [\mathbf{x}, \mathbf{h}] \in \mathbb{R}^{N \times (3+d)}$, with $\mathbf{x} \in \mathbb{R}^{N \times 3}$ being the N atom's 3D coordinates and $\mathbf{h} \in \mathbb{R}^{N \times d}$ as the atom features such as types. A latent diffusion architecture [Rombach et al., 2022, Xu et al., 2023] consists of a VAE and a diffusion model, and are trained consecutively. Particularly, the geometric latent diffusion model [Xu et al., 2023] uses the encoder of the VAE to project discrete molecules to a continuous latent space, on which the diffusion

model is then trained. Denote the encoder as $\mathcal{E}$ and the latent variable by $\mathbf{z} \in \mathbb{R}^{N\times(3+d_z)}$, then $[\mathbf{z}_{\mathbf{x},0}, \mathbf{z}_{\mathbf{h},0}] = \mathcal{E}([\mathbf{x}, \mathbf{h}])$, with $\mathbf{z}_{\mathbf{h},0} \in \mathbb{R}^{N\times d_z}$ and $d_z < d$. Let $\mathbf{z}_t = [\mathbf{z}_{\mathbf{x},t}, \mathbf{z}_{\mathbf{h},t}]$, the latent forward diffusion process and reverse denoising process are defined as:

$$q(\mathbf{z}_t \mid \mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t; \sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I}) \tag{5.4}$$

$$p_\theta(\mathbf{z}_{t-1} \mid \mathbf{z}_t) = \mathcal{N}(\mathbf{z}_{t-1}; \mu_\theta(\mathbf{z}_t, t), \Sigma_\theta(\mathbf{z}_t, t)) \tag{5.5}$$

We denote the decoder of the VAE as $\mathcal{D}$, which maps $\mathbf{z}_0$ back to the original molecular space, such that $\mathcal{D}([\mathbf{z}_{\mathbf{x},0}, \mathbf{z}_{\mathbf{h},0}]) = [\mathbf{x}, \mathbf{h}] \in \mathbb{R}^{N\times(3+d)}$.

The encoder $\mathcal{E}$ and the decoder $\mathcal{D}$ are parameterized with an equivariant graph neural network (EGNN) [Satorras et al., 2021] to translate between discrete molecular data and latent variables, such that the atom types are invariant and the positions are equivariant to transformations as follows:

$$R\mathbf{z}_{\mathbf{x},t} + T, \mathbf{z}_{\mathbf{h},t} = \mathcal{E}(R\mathbf{x}_t + T, \mathbf{h}_t) \qquad R\mathbf{x}_t + T, \mathbf{h}_t = \mathcal{D}(R\mathbf{z}_{\mathbf{x},t} + T, \mathbf{z}_{\mathbf{h},t}) \tag{5.6}$$

for any rotation matrix R and translation matrix T, where $\mathbf{z}_{\mathbf{x},t} \in \mathbb{R}^{N\times 3}$ are required to satisfy zero center gravity and have zero-mean over N atoms for each position. In addition, the latent diffusion model is also parameterized by EGNN such that transitions between each timestep in the denoising process also respect the same characteristics.

GFN2-xTB Method According to the laws of physics and thermodynamics, matter such as electrons, atoms, and molecules, interacts with matter inherently to reach configurations with lower potential energies for better stability. To formulate this as a molecular geometry optimization problem, let $h_1, \cdots, h_N$ be the N atoms in a given molecule and $\mathbf{x_1}, \cdots, \mathbf{x_N} \in \mathbb{R}^3$ be their corresponding coordinates in the 3D space, then each atom h_i is subject to a force

$$\mathbf{f_i}(\mathcal{G}) = -\frac{\partial E_p\left(\mathbf{x_1}, \cdots, \mathbf{x_N} \mid h_1, \cdots, h_N\right)}{\partial \mathbf{x_i}}, \forall i \in [N] \tag{5.7}$$

where E_p represents the potential energy of the conformation. The force here manifests valid physical interpretations: an atom is pushed accordingly by the exerted force until the force reduces to zero and an equilibrium is achieved. One necessary (but not sufficient) condition for a stable molecular geometry is that all forces on the atoms should be (close to) zero, i.e., $\forall i \in [N], \mathbf{f_i}(\mathcal{G}) = 0$.

However, the exact mathematical potential energy evaluation, i.e., the solution to the Schrödinger equation, is still a black box to us [Cao et al.]. Over the years, different levels of theories and methods have been developed to evaluate the potential energy, such as force field (FF), semi-empirical methods (e.g., xTB), and density functional theory (DFT) methods (e.g., B3LYP/6-31G(2df,p)). The methods are listed in order of increased accuracy and cost. After trading-off between accuracy and efficiency within a feasible computation cost, we selected GFN2-xTB, a more recent and advanced semi-empirical method [Bannwarth et al., 2019], to calculate the forces of the generated molecular geometry in the diffusion process.

GFN stands for, respectively, **g**eometry optimization, vibrational **f**requencies, and **n**on-covalent interactions. xTB refers to e**x**tended **t**ight **b**inding, and 2 refers to the version. In the GFN2-xTB

method, the total energy expression is given by Bannwarth et al., 2019]

$$E_{\text{GFN2-xTB}} = E_{\text{rep}} + E_{\text{disp}} + E_{\text{EHT}} + E_{\text{IES+IXC}} + E_{\text{AES}} + E_{\text{AXC}} + G_{\text{Fermi}}$$

, where E_{rep} is the repulsive energy contribution from short-range interactions, E_{disp} is the dispersion energy contribution from long-range interactions, E_{EHT} is the energy contribution from the extended Hückel theory (EHT), $E_{\text{IES+IXC}}$ is the isotropic electrostatic (IES) energy contribution and the isotropic exchange-correlation (IXC) energy contribution, E_{AES} is the anisotropic electrostatic (AES) energy contribution, E_{AXC} is the anisotropic exchange-correlation (AXC) energy contribution, and G_{Fermi} is the entropic contribution of an electronic free energy at finite electronic temperature T_{el} due to Fermi smearing.

Its accuracy and efficiency come strictly from the element-specific and global parameters for all elements up to radon (Z = 86) Bannwarth et al. [2019], hence the semi-empiricism. The pre-computed tight-binding parameters and empirical corrections are utilized to approximate the electronic structure and calculate energy contributions efficiently.

5.4 Methodology

Given molecular data represented in the form of a point cloud $\mathcal{G} = [\mathbf{x}, \mathbf{h}] \in \mathbb{R}^{N\times(3+d)}$, with $\mathbf{x} \in \mathbb{R}^{N\times 3}$ being the N atom's 3D coordinates and $\mathbf{h} \in \mathbb{R}^{N\times d}$ as the atom features such as atom types, e.g., atomic numbers, we denote the generated molecule by the diffusion model as $\mathcal{G}_0$. We introduce a training-free guided denoising process by incorporating the GFN2-xTB method at inference time to minimize the net force acting on each atom and optimize the molecular geometry towards a more stable atom configuration. We denote the molecule generated by such process as $\mathcal{G}'$ and we aim to achieve $\frac{1}{N}\sum_i^N \mathbf{f_i}(\mathcal{G}'_0) < \frac{1}{N}\sum_i^N \mathbf{f_i}(\mathcal{G}_0)$ for $i \in [N]$, with $\mathbf{f_i}(\mathcal{G})$ defined as Eq. 5.7.

Up next, we will first describe how to perform neural regressor guidance [Vignac et al., 2022, Han et al., 2023, Bao et al., 2022], then introduce how to use a non-differentiable oracle in place of a differentiable regressor to obtain the gradient for diffusion guidance. We also introduce a bi-level optimization framework that incorporates both guidance from the neural regressor and the non-differentiable oracle.

5.4.1 Guidance From Neural Regressor

The goal of neural guidance is to direct the denoising process towards a target property value y. [Dhariwal and Nichol, 2021] proposed a way to modify the denoising process to achieve conditional generation with an unconditional diffusion model, with a scalar s controlling the guidance strength:

$$\boldsymbol{x}_{t-1} \sim \mathcal{N}\left(\frac{1}{1-\beta_t}\left(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t^2}}\epsilon_\theta(\boldsymbol{x}_t, t)\right) + s\rho_t^2\nabla_{\boldsymbol{x}_t}\log p_\phi(y|\boldsymbol{x}_t), \rho_t^2\mathbf{I}\right) \tag{5.8}$$

In their case, $p_\phi(y|\boldsymbol{x}_t)$ is parameterized by a classifier and y is a categorical label, where the modification $s\rho_t^2\nabla_{\boldsymbol{x}_t}\log p_\phi(y|\boldsymbol{x}_t)$ shifts the mean of the sampling distribution to provide guidance.

However, we are interested in the case of $y \in \mathbb{R}$. Let $f_\eta : \mathcal{G} \to \mathbb{R}$ be the neural regressor for the property of interest, where we follow [Vignac et al., 2022] and assume $y|\boldsymbol{x}_t \sim \mathcal{N}(f_\eta(\boldsymbol{x}_t), \sigma_\eta^2 \mathbf{I})$ and $\sigma_\eta^2 = 1$ for simplicity, we can estimate that:

$$\nabla_{\boldsymbol{x}_t} \log p_\phi(y \mid \boldsymbol{x}_t) \propto -\nabla_{\boldsymbol{x}_t} \|y - f_\eta(\boldsymbol{x}_t)\|_2^2 = -\nabla_{\boldsymbol{x}_t} \mathcal{L}(y, f_\eta(\boldsymbol{x}_t)) \tag{5.9}$$

where $\mathcal{L}(y, f_\eta(\boldsymbol{x}_t))$ is the MSE between the target and predicted value by the regressor $f_\eta(\cdot)$.

However, in the early stage of the denoising process, $\boldsymbol{x}_t = \alpha_t \boldsymbol{x}_0 + \sigma_t \epsilon$ might not be informative enough to predict y as it consists mostly of Gaussian noise. For more effective prediction during the denoising process, we can estimate the denoised version of $\boldsymbol{x}_t$ as [Kawar et al., 2022, Song et al., 2020a]:

$$\hat{\boldsymbol{x}}_0 = \frac{\boldsymbol{x}_t - \sqrt{1-\alpha_t}\epsilon_\theta(\boldsymbol{x}_t, t)}{\sqrt{\alpha_t}} \tag{5.10}$$

And $\hat{\boldsymbol{x}}_0$ can be used in place of $\boldsymbol{x}_t$ when calculating the guidance, such that

$$\nabla_{\boldsymbol{x}_t} \log p_\phi(y|\boldsymbol{x}_t) \approx \nabla_{\boldsymbol{x}_t} \log p_\phi(y|\hat{\boldsymbol{x}}_0) \tag{5.11}$$

5.4.2 Guidance From a Non-Differentiable Oracle

In this section, we aim to tackle a more challenging problem where the guidance is specified by a non-differentiable oracle. Since we adopted the latent diffusion architecture such that the diffusion model is trained on the continuous latent space encoded by VAE, we change our notation $\boldsymbol{x}_t$ for each state in the diffusion model in Section 5.4.1 to $\mathbf{z}_t$ as in Section 5.3. To precisely formulate our problem, which aims to refine the geometry and improve the stability of the generated molecules by minimizing the net force acting on each atom, our non-differentiable function $\mathbf{f}$ for guidance can be defined as the following:

$$\mathbf{f}(\mathcal{G}) = \frac{1}{N} \sum_i^N \mathbf{f_i}(\mathcal{G}) \tag{5.12}$$

where the molecular graph $\mathcal{G}$ is obtained by decoding the continuous latent variable through the decoder $\mathcal{G} = \mathcal{D}(\hat{\mathbf{z}}_0)$, $\hat{\mathbf{z}}_0$ is estimated by Eq. 5.10 and $\mathbf{f_i}$ is our non-differentiable oracle defined in Eq. 5.7 that yields the net force on each atom i, and our target value y in this case is 0.

When $\mathbf{f}$ is non-differentiable, we can no longer conduct analytical differentiation and plug it into Eq. 5.9 to obtain the gradient as guidance. Thus, we use numerical differentiation in place of it. For ease of discussion, we define the one-step estimation of the original datapoint $\mathbf{z}_0$, introduced in Eq 5.10, as the function $t_0(\cdot)$:

$$\hat{\mathbf{z}}_0 = t_0(\mathbf{z}_t) = \frac{\mathbf{z}_t - \sqrt{1-\alpha_t}\epsilon_\theta(\mathbf{z}_t, t)}{\sqrt{\alpha_t}} \tag{5.13}$$

Recall that $\mathcal{L}(y, \mathbf{f}(\mathcal{G}))$ is the MSE loss function, $\mathbf{f} : \mathcal{G} \to \mathbb{R}$ is the non-differentiable oracle, and $\mathcal{D}$ is the decoder. We denote $\mathcal{F}$ as the composition $\mathbf{f} \circ \mathcal{D} \circ t_0$, the gradient is estimated numerically by

$$\hat{\nabla}_{\mathbf{z}_t} \log p_\phi(y|\mathbf{z}_t) \propto -\nabla_{\mathcal{F}(\mathbf{z}_t)} \mathcal{L}(y, \mathcal{F}(\mathbf{z}_t)) \nabla_{\mathbf{z}_t} \mathcal{F}(\mathbf{z}_t) \tag{5.14}$$

$$\approx -\nabla_{\mathcal{F}(\mathbf{z}_t)}\mathcal{L}(y, \mathcal{F}(\mathbf{z}_t))\hat{\nabla}_{z_t}\mathcal{F}(\mathbf{z}_t) \tag{5.15}$$

where each element at location i-th row and j-th column in $\hat{\nabla}_{z_t}\mathcal{F}(\mathbf{z}_t)$ is estimated with

$$\hat{\nabla}^{i,j}_{z_t}\mathcal{F}(\mathbf{z}_t) = \lim_{\zeta \to 0} \frac{\mathcal{F}([\mathbf{z}_{\mathbf{x},t} + \zeta e^{i,j}_{N\times 3}, \mathbf{z}_{\mathbf{h},t}]) - \mathcal{F}([\mathbf{z}_{\mathbf{x},t} - \zeta e^{i,j}_{N\times 3}, \mathbf{z}_{\mathbf{h},t}])}{2\zeta}$$

and $e^{i,j}_{N\times 3}$ represents a N by 3 zero matrix with 1 in location i-th row and j-th column.

This approximation is possible because $\mathbf{z}_t$ is continuous after the projection of the VAE encoder. The formulation allows for guidance from a non-differentiable oracle such as quantum chemical method GFN2-xTB [Bannwarth et al., 2019]. Estimating the gradient directly from an expert oracle eliminates the need to train additional neural regressors for properties such as the force on each atom, which comes with approximation error [Gasteiger et al., 2020]. In addition, these regressors are often trained on stable molecules from QM9 instead of the potentially unstable molecules as byproducts of the denoising process, making them dubious options for property guidance.

However, directly adding $\pm\zeta\mathbf{1}^{i,j}_{N\times 3}$ to $\mathbf{z}_{\mathbf{x},t}$ would break the equivariance requirement in Eq. 5.6 on the latent variables, as it shifts the mean of the coordinates by ζ. To maintain zero center gravity of input to the $\mathcal{F}(\cdot)$, we construct a perturbation matrix $\mathbf{U} \in \mathbb{R}^{N\times 3}$ where each element is sampled from $\mathcal{N}(0, 1)$, and apply Simultaneous Perturbation Stochastic Approximation (SPSA) [Spall, 1992, Nesterov and Spokoiny, 2017, Malladi et al., 2023] to estimate the gradient; thus, Eq. 5.15 becomes

$$\hat{\nabla}_{\mathbf{z}_t} \log p_\phi(y|\mathbf{z}_t) \propto -\nabla_{\mathcal{F}(\mathbf{z}_t)}\mathcal{L}(y, \mathcal{F}(\mathbf{z}_t)) \frac{\mathcal{F}([\mathbf{z}_{\mathbf{x},t} + \zeta\mathbf{U}, \mathbf{z}_{\mathbf{h},t}]) - \mathcal{F}([\mathbf{z}_{\mathbf{x},t} + \zeta\mathbf{U}, \mathbf{z}_{\mathbf{h},t}])}{2\zeta}\mathbf{U} \tag{5.16}$$

where ζ is a small perturbation scale (e.g., 10^{-6}). The perturbed representations $[\mathbf{z}_{\mathbf{x},t} \pm \zeta\mathbf{U}, \mathbf{z}_{\mathbf{h},t}]$ approximately achieves zero center gravity as $\frac{1}{3N}\sum_{j=1}^{3}\sum_{i=1}^{N}\mathbf{U}_{ij} \approx 0$. Note that, unlike neural regressor guidance, we only add guidance to the positions, i.e., $\mathbf{z}_{\mathbf{x},t}$, and apply no gradient to the atom types, i.e., $\mathbf{z}_{\mathbf{h},t}$. This is because the force definition (Eq. 5.7) is only physically grounded when the set of atoms stays constant, i.e., no matter/mass is created from or reduced to void. According to Einstein's mass-energy equivalence ($E = mc^2$), any change in atom type would change the mass and create a tremendous potential energy change, which is not within the topics of this work.

5.4.3 Combine Guidance from Neural Regressor and Non-Differentiable Oracle

Apart from guided generation conditioned on one single property with a regressor neural network (e.g., static polarizability α) or a non-differentiable oracle (e.g., force), we can perform multi-property optimization with the guidance specified by both the differentiable regressor and the non-differentiable oracle. We encapsulate it under the framework of *bi-level optimization*. Formally, recall that f_η represents the regressor and $\mathcal{F} = \mathbf{f}\circ\mathcal{D}\circ t_0$ is the composition of the non-differentiable oracle $\mathbf{f}$, the VAE decoder is denoted as$\mathcal{D}$, the multi-property optimization can be formulated as:

$$\min_{\mathbf{z}_{x,t}} \quad \mathcal{F}([\mathbf{z}_{x,t}, \mathbf{z}^*_{h,t}]) \tag{5.17}$$

Algorithm 6 Denoising Diffusion Guidance with Non-Differentiable Oracle

Input: A latent diffusion model θ, a VAE decoder $\mathcal{D}$, a non-differentiable composition function $\mathcal{F}$, guidance scale s, SPSA perturbation ζ, and the target property y
$\mathbf{z}_T \leftarrow \mathcal{N}(0, \mathbf{I})$
for t from $T-1$ to 0 **do**
  $\mu_{t-1} \leftarrow \frac{1}{1-\beta_t}(\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t^2}}\epsilon_\theta(\mathbf{z}_t, t))$
  Sample $\mathbf{U}_i \sim \mathcal{N}(0,1)$ for each i-th element of $\mathbf{U}$
  $g_{t-1} \propto -\nabla_{\mathcal{F}(\mathbf{z}_t)}\mathcal{L}(y, \mathcal{F}(\mathbf{z}_t))\frac{\mathcal{F}(\mathbf{z}_t+\zeta\mathbf{U})-\mathcal{F}(\mathbf{z}_t-\zeta\mathbf{U})}{2\zeta}\mathbf{U}$ ▷ Gradient Estimation
  $\mathbf{z}_{t-1} \leftarrow \mathcal{N}(\mu_{t-1} + s\rho_t^2 g_{t-1}, \rho_t^2\mathbf{I})$ ▷ Guided Sampling
end for
$\mathbf{x}, \mathbf{h} \leftarrow \mathcal{D}(\mathbf{z}_0)$
return $\mathbf{x}, \mathbf{h}$

$$\text{s.t.} \quad [\mathbf{z}_{x,t}^*, \mathbf{z}_{h,t}^*] = \underset{[\mathbf{z}_{x,t}, \mathbf{z}_{h,t}]}{\arg\min} \| f_\eta \circ \mathcal{D}([\mathbf{z}_{x,t}, \mathbf{z}_{h,t}]) - y \|^2 \tag{5.18}$$

where y is the target property value (e.g., we hope the static polarizability $\alpha \to y$ after optimization). As detailed in Algorithm 6, we apply self-recurrence steps (Eq. **??**) to optimize the property scores given by f_η, and obtain the optimal atom positions $\mathbf{z}_{x,t}^*$ and features $\mathbf{z}_{h,t}^*$, as shown in Eq. 5.18; we then optimize atom stability given by $\mathcal{F}$ by fixing $\mathbf{z}_{h,t}^*$ and initializing $\mathbf{z}_{x,t}$ in Eq. 5.17 as $\mathbf{z}_{x,t}^*$ from Eq. 5.18.

Chemistry Intuition The task of generating molecules with target properties is closely related to the field of rational chemical design such as drug design. Domain knowledge is widely used to guide the design process. For example, benzene rings that contain oxygen (e.g., dioxins) and sulfur (e.g., thiophenes) are typically known to be toxic and carcinogenic. Therefore, when designing drugs, it is typical to stay away from such species prior to performing further property analysis such as bind-affinity to a specific protein or DNA. In such design processes, the overall structures are examined by experts before further target property analysis.

The conflict behind generating stable molecules with target properties is that properties are closely related to both atom types $\mathbf{h}$ and atom coordinates $\mathbf{x}$. We are mostly concerned with the properties of a stable molecule with an optimized geometry but it is not known *a priori* given the atom types. In the diffusion process to generate molecules, the atom types are generated and stabilized during earlier steps, and their coordinates are further optimized in later steps. Since our target is to generate stable molecules with desired properties, there is a conflict between stability-first and property-first in the generation process.

The *bi-level* property optimization proposed here takes inspiration from the rational design process: higher-level overall structure before lower-level property detail analysis. An overall molecular structure is generated with desired properties in earlier steps, where the diffusion model is considered as chemistry/biology/etc. experts performing the initial structure-property analysis. In the later steps, the geometries are further optimized.

5.5 Experiments

In this section, we present two sets of experiments: 1) experiment on QM9 [Ramakrishnan et al., 2014] and GEOM [Axelrod and Gómez-Bombarelli] dataset to check if using an estimated gradient of the net force on each atom for reverse diffusion sampling can increase the rate of stability among the generated molecules; 2) combining the estimated gradient from the non-differentiable oracle and a gradient from a differentiable property predictor, we can obtain molecules with improved properties for both criteria. In the following, we will briefly introduce the datasets, properties, baselines, and our evaluation metrics.

Dataset The models in our experiment are trained on the QM9 dataset [Ramakrishnan et al., 2014] and the GEOM dataset [Axelrod and Gómez-Bombarelli]. The QM9 dataset [Ramakrishnan et al., 2014] is a catalog with 133,885 small organic molecules consisting of up to nine heavy (non-hydrogen) atoms. The dataset also includes quantum chemical properties calculated by DFT. The Geometric Ensemble Of Molecules (GEOM) dataset [Axelrod and Gómez-Bombarelli] includes 450K molecules with up to 91 heavy atoms (on average, 24.9), where 37 million molecular conformations are generated and reported with their geometries, energies, and statistical weight.
Guidance Property We study guided generation for optimized geometries on both QM9 and GEOM datasets. For neural regressor guidance, we evaluate on QM9 as there are no property labels for the regressor to be trained on GEOM. We consider the following 6 properties on QM9:

- the norm of static polarizability (α),
- the norm of dipole moment (μ),
- heat capacity at room temperature (C_v),
- the energy of the electron in the highest occupied molecular orbital (ϵ_{HOMO}),
- the energy of the electron in the lowest unoccupied molecular orbital (ϵ_{LUMO}),
- the HOMO-LUMO energy gap ($\Delta\epsilon$).

Baseline We apply our method on GeoLDM, which shows improved performance compared with EDM, and compare to different versions of EDM and GeoLDM. For non-differentiable guidance on forces, we compare our method to unconditional EDM [Hoogeboom et al., 2022] and GeoLDM [Xu et al., 2023], as there are no available ground-truth forces to train a conditional model. For neural regressor guidance and combined guidance, we choose conditionally trained EDM and GeoLDM as the baselines.
Evaluation Metric For non-differentiable guidance on forces, We report the Root Mean Square (MSE) of the forces calculated using xTB as the evaluation metric. We also report results computed using DFT at the B3LYP/6-31G(2df,p) level of theory for QM9, a method that is more accurate but exponentially more demanding in computation than GFN2-xTB. For differentiable neural guidance on the 6 properties ($\alpha, \mu, C_v, \epsilon_{\text{HOMO}}, \epsilon_{\text{LUMO}}, \Delta\epsilon$), we follow Xu et al. [2023] and use the Mean Absolute Error (MAE) between the target property y and the predicted value $\hat{y}$ by the regressor of the generated molecule as the metric.
Computation & Implementation The acceleration hardware we used is an NVIDIA RTX A6000 with 48 GiB RAM and the AMD EPYC 7513 32-Core Processor, where 1 diffusion step with xTB guidance takes up ~ 20 seconds on QM9 for 300 molecules. We use pre-trained EDM and GeoLDM provided by the authors as our baselines.

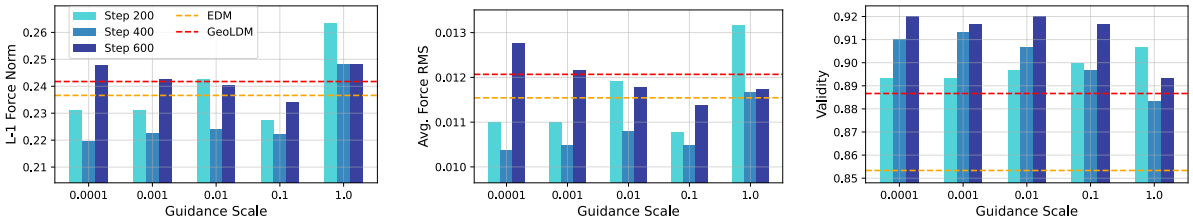


Figure 5.3: L-1 norm (*left*) and average RMS (*middle*) of forces and validity (*right*) of 300 generated molecules on QM9 with an unconditional GeoLDM guided by xTB, across different guidance steps (200, 400 and 600 steps) and scales (0.0001, 0.001, 0.01, 0.1, and 1) and we show their performances on l1 norm and RMS of forces on all atoms as well as general validity.

5.5.1 Non-Differentiable Oracle Guidance

The first question we want to investigate is at which point in the denoising process should we start adding in the force regularization. We conduct experiments with varying numbers of timestep where the force guidance is conducted and represent our results in Figure 5.3, where step number t stands for adding guidance from step t to 0. It is observed that over different guidance steps and scales, our non-differentiable oracle guidance can achieve lower forces while generating more valid molecules. Specifically, when the guidance steps begin from the last 400 steps, our method performs the best in terms of low RMS and L-1 norm net forces and high validity rate, which is consistent with the findings of Han et al. [2024].

To gain further insights, we also report the numerical results on 400 guidance steps in Table 5.1 computed by the xTB method as well as in Table 5.3 computed using DFT, a more accurate but computationally costly method. As evaluated by xTB method, which is also the method we use to estimate the gradient for guidance, the force-guided denoising process achieves lower force RMS with all scales and a higher level of stability than the GeoLDM method with all but one scale. The trend indicates that scales of 0.0001 and 0.001 generally achieve lower force RMS and high validity. Our sampling method also outperforms the force RMS and validity results by sampling from EDM. This shows the effectiveness of our method. The evaluation by the DFT method shows similar improvements from our baseline GeoLDM method with validity going from 93.67% to 96.67% and force RMS going down from 0.0061 to 0.0051, even though our sampling method is not guided by it due to computational constraints. This shows that guidance by xTB has generalizable improvements in the quality of the output molecules, even though it is considered a less accurate quantum mechanic method. A similar trend can observed on the GEOM dataset in Table 5.2. Because the GEOM dataset includes molecules that can be significantly larger than in QM9, our results are averaged over 100 molecules. With force guidance, validity improves from 50% of the GeoLDM baseline to 56% and force RMS reduces from 0.0427 of the baseline to 0.0398.

5.5.2 Neural Regressor Guidance

In this section, we present the results of multiple differentiable regressor-guided denoising processes. This is in preparation for combining differentiable neural guidance with the non-

Guidance Scale	0.0001	0.001	0.01	0.1	1.0
Force RMS	**0.0104**	0.0105	0.0108	0.0105	0.0117
Validity	91.00%	**91.33%**	90.67%	89.67%	88.33%
EDM	Force RMS 0.0115 / Validity 85.33%				
GeoLDM	Force RMS 0.0121 / Validity 88.67%				

Table 5.1: Force RMS and validity over 300 generated molecules on QM9 using xTB calculation.

Guidance Scale	0.0001	0.001	0.01	0.1	1.0
Force RMS	**0.0398**	0.0432	0.0451	0.0417	0.0477
Validity	47.0%	48.0%	51.0%	**56.0%**	48.0%
EDM	Force RMS 0.0787 / Validity 45.0%				
GeoLDM	Force RMS 0.0427 / Validity 50.0%				

Table 5.2: Force RMS and validity over 100 generated molecules on GEOM using xTB calculation

differentiable oracle guidance in the next section. The regressors are used to predict the quantum mechanical properties of the molecules.

We present our neural regressor-guided results in Table 5.4, where we apply guidance at each diffusion step t. When comparing to our baseline GeoLDM, Our method achieves the best MAE in terms of α, μ, C_v and slightly worse on the rest 3 properties. Overall, higher scales have yielded better results but a scale of size 40 or 50 only yields invalid results. Recent work Equivariant Energy Guided SDE (EEGSDE) by Bao et al. [2022] and MuDM by Han et al. [2023] also studied similar property-guided denoising processes for molecule generation and adopted an SDE-based diffusion model. We include their results in Table 5.4 as well. Notice that we are unfortunately not able to recover their results, possibly due to implementation differences as we are working with a DDPM-styled latent diffusion model. We chose the latent diffusion model initially because it would provide a continuous representation of the molecules, this is particularly useful because atom types are usually represented as discrete. However, we discovered that applying guidance on the atom types can cause the calculation to explode, so we constrained the force guidance strictly to the dimensions of the coordinate. This way, applying the diffusion model directly to this problem without the VAE would also work. Trying a non-latent diffusion architecture would be a promising future work.

5.5.3 Combined Guidance

For this section, we want to generate molecules with a lower net force on each atom conditioned on a target property value. We present our results on multi-objective generation in Table 5.5 and 5.6. Particularly, Table 5.5 evaluates the mean absolute error between the generated molecules' property values and each target property, and Table 5.6 evaluates the L1 norm of the net force on generated molecules. We experimented with two types of guidance methods, one applies xTB guidance to a conditional GeoLDM every $\{1, 3, 5\}$ step(s), and another applies both neural regressor guidance and non-differentiable xTB guidance at each step to an unconditioned GeoLDM.

Guidance Scale	0.0001	0.001	0.01	0.1	1.0
Force RMS	**0.0051***	0.0052	0.0054	0.0052	0.0061
Validity	96.33%	**96.67%***	96.33%	96.0%	94.0%
EDM	Force RMS 0.0051 / Validity 89.33%				
GeoLDM	Force RMS 0.0061 / Validity 93.67%				

Table 5.3: Force RMS and validity of 300 generated molecules on QM9 using DFT calculation. $*$ and **bold** denote the overall best and the best within different scales, respectively.

Property		α	μ	C_v	ϵ_{HOMO}	ϵ_{LUMO}	$\Delta\epsilon$
Units		Bohr^3	D	$\frac{\text{cal}}{\text{mol}}$K	meV	meV	meV
GeoLDM		6.5535	1.3873	2.8644	0.6475	1.2095	1.2298
Conditional EDM		3.0564	1.1514	1.1081	0.3737*	0.5997*	0.6389*
Conditional GeoLDM		3.4111	1.2298	1.2194	0.4278	0.6579	0.7536
EEGSDE		2.50	0.777	0.941	0.302	0.447	0.487
MuDM		0.43	0.333	0.290	0.072	0.133	0.085
Ours	10^{-4}	5.7699	1.5315	3.0478	0.6601	1.2369	1.1810
	10^{-3}	5.7886	1.5237	3.0603	0.6610	1.2374	1.1789
	10^{-2}	5.7690	1.5402	3.0680	0.6540	1.2273	1.1718
	10^{-1}	5.4502	1.5309	2.8668	0.6458	1.2277	1.1663
	1	4.2645	1.3900	2.3090	0.6429	1.1830	1.1080
	2	3.9743	1.3950	2.0536	0.6317	1.1180	1.0893
	5	3.7323	1.3616	1.5969	0.6498	1.1254	1.0325
	10	3.4368	1.1631	1.4279	0.6024	1.0732	0.9786
	20	**2.7576***	**0.9831***	1.1085	**0.5551**	0.9166	0.9534
	25	2.8115	✗	1.1184	0.5662	0.8953	0.9634
	30	✗	✗	**1.0640***	0.5644	**0.8541**	**0.8949**
	40	✗	✗	✗	✗	✗	✗

Table 5.4: MAE of conditional EDM, conditional GeoLDM, and our regressor guided generation with various guidance scales. $*$ and **bold** denote the overall best and the best within different scales, respectively. ✗ marks the settings that caused NaN outputs, and we omit the results with scale ≥ 40 as the guidance fails for all properties.

Overall, xTB-guided conditional GeoLDM performs better than the Bi-Level framework in Table 5.5. This is probably due to the weak performance of this particular implementation of neural regressor guidance as seen in the last section. The xTB guidance with conditional GeoLDM

achieves the best performance in α (10.0% improvement) and ϵ_{HOMO} (5.0% improvement) compared to all other methods. Particularly, it outperforms conditional GeoLDM for *all* properties, signaling that an improved geometry could also improve property compliance. The bi-level framework beats the conditional GeoLDM in α and μ, where we see improvements of 3.0% and 2.5% respectively, but performs worse in other properties.

For Table 5.6, as each property requires training separate conditional model, we present the results on average net force on the atom for each model. We notice that conditional GeoLDM with xTB guidance significantly outperforms conditional GeoLDM in all property models except ϵ_{HOMO} and Δ_ϵ. For those two properties, the results are only slightly worse. The lowest net force is from the Bi-Level framework for every property. Another interesting finding is that conditional EDM has a lower average net force on each atom than the conditional GeoLDM for all these models.

For conditional GeoLDM with xTB guidance, we see that applying xTB every 3 steps seems to be the optimum as opposed to every step or every five steps. It achieves the lower MAE of properties in 4 out of 6 properties and lower l1 norm of force in also 4 out of 6 property models. Achieving target property values and minimizing net force on each atom could potentially be competing objectives, as more frequent xTB guidance could move the generated molecules away from the target properties.

Property Units		α Bohr^3	μ D	C_v $\frac{\text{cal}}{\text{mol}}$K	ϵ_{HOMO} meV	ϵ_{LUMO} meV	$\Delta\epsilon$ meV
Conditional EDM		3.0564	1.1514*	1.1081*	0.3737	0.5997*	0.6389*
Conditional GeoLDM		3.4111	1.2298	1.2194	0.4278	0.6579	0.7536
Unconditional GeoLDM		6.5535	1.3873	2.8644	0.6475	1.2095	1.2298
Bi-Level		3.2762	1.1942	✗	0.6132	0.9617	0.9777
Conditional GeoLDM with xTB Guidance	Every Step	3.0394*	1.1842	1.2341	0.3898	0.6431	0.6973
	Every 2 Steps	**2.5654**	**1.1618**	1.2607	0.3878	**0.6266**	**0.6735**
	Every 3 Steps	2.8449	1.1709	**1.2053**	**0.3577***	0.6307	0.7037

Table 5.5: MAE of properties of 300 sampled molecules from conditional GeoLDM with xTB guidance and bi-level optimization framework. $*$ and **bold** denote the overall best and the best within different scales, respectively. ✗ marks the settings that caused NaN outputs.

5.5.4 Study on Skipped Guidance

As the xTB runs on CPU and is time-consuming, we propose a *skip-step* acceleration method for xTB-guided optimization. Specifically, let the skip-step be k, we only calculate gradients from xTB every k step and use historical gradients for the steps with no xTB calculation. The results are shown in Table 5.7. For skip steps = 2 and 3, we form combinations of guidance steps ($t = 200, 400, 600$) and scales ($s = 0.001, 0.0001$), we measure the performance in the L-1 norm of force, avg. RMS force, and validity. We show that the results are competitive with our previous method without skip-step. Particularly, the highest validity 94.00%, generated by the hyperparameter combinations skip step = 2, guidance step = 400, and guidance scale = 0.0001, is

Property		α	μ	C_v	ϵ_{HOMO}	ϵ_{LUMO}	$\Delta\epsilon$
Conditional EDM		0.2536	0.2465	0.2484*	0.2330	0.2432	0.2366
Conditional GeoLDM		0.3606	0.3472	0.3728	0.3437	0.3972	0.2907
Bi-Level		0.2167*	0.2191*	✗	0.2061*	0.2208*	0.2089*
Conditional GeoLDM with xTB Guidance	Every Step	**0.3228**	0.3743	0.3561	0.3506	0.3470	**0.2993**
	Every 3 Steps	0.3357	**0.3243**	**0.3278**	**0.3466**	**0.3268**	0.3064
	Every 5 Steps	0.3418	0.3292	0.3462	0.3580	0.3471	0.3032

Table 5.6: L1 norm of force of 300 sampled molecules from conditional GeoLDM with xTB guidance and bi-level optimization framework. $*$ and **bold** denote the overall best and the best within different scales, respectively. ✗ marks the settings that caused NaN outputs.

Skip Step	Guidance Step	Guidance Scale	L1 Force Norm	Force RMS	Validity
2	200	0.0001	0.2374	0.0114	90.33%
		0.001	0.2306	0.0108	90.0%
	400	0.0001	0.2283	0.0110	**94.0%***
		0.001	**0.2273***	0.0109	93.33%
	600	0.0001	0.2347	0.0113	92.67%
		0.001	0.2426	0.0119	93.67%
3	200	0.0001	0.2274	**0.0107***	88.67%
		0.001	**0.2273***	**0.0107***	88.67%
	400	0.0001	0.2293	0.0109	91.33%
		0.001	0.2297	0.0110	91.33%
	600	0.0001	0.2401	0.0119	92.0%
		0.001	0.2379	0.0116	92.33%

Table 5.7: L-1 norm of force, avg. force RMS, and validity from 300 generated molecules sampled from the QM9 dataset using GeoLDM with xTB guidance and various skip-step acceleration schedules. $*$ and **bold** denote the overall best and the best within different scales, respectively. ✗ marks the situation where the guidance collapses.

even higher than our best validity without skipping steps from Table 5.1 at 91.33%. The highest force RMS from the skip step approach is 0.0107 is only slightly higher than the best results in Table 5.1.

5.5.5 Conclusion & Limitation

In this work, we study conditional generation for 3D diffusion models on molecules, where we steer the generation process of a diffusion model trained unconditionally with a non-differentiable chemistry oracle. By estimating the guidance gradient with an analytic solution while respecting the equivariance requirements for 3D diffusion models, we can generate molecules with low net

force and high validity. Further experiments on incorporating both non-differentiable guidance and neural guidance demonstrate the effectiveness of our method. However, since the xTB runs on CPU and neural guidance requires gradients that can take up a lot of GPU resources, our method can be ineffective where the computational resources are insufficient, and future works can explore the possibility of speeding up the guidance steps while reducing computational costs and maintains good performance.

Chapter 6

Conclusion

In this thesis, we present a collection of work on applying generative models to the domain of generation of discrete structured data such as program synthesis and drug discovery as well as a study on the mechanism of VAE, a type of generative model.

In Section 2, we studied the application of autoregressive models for the task of program synthesis, particularly, parsing geometric images into context-free grammar programs. Due to the non-differentiable renderer used to evaluate the programs in the generation pipeline, we use REINFORCE as our objective. The biggest challenge of the problem is scaling up to a search space exponential to the program length. For that, we proposed three techniques to overcome the challenge: 1) a grammar-encoded tree LSTM to limit the output space to only the valid ones; 2) entropy regularization for better exploration; 3) stochastic beam search during training that achieves sampling without replacement from the CFG syntax tree. These techniques enable effective identification of the correct program in the large search space and we show dramatically improved results on a synthetic data set and a 2D CAD dataset in comparison to the vanilla REINFORCE approach.

In Section 3, we studied the manifold recovery ability of VAEs, and found that for non-linear data, VAEs are not guaranteed to recover the manifold or the data density on the manifold. However, in further experiments, we found that multi-stages of VAEs trained sequentially can improve data manifold recovery. Thus, in Section 4, we apply the findings to discrete structured data for the problem of molecule generation. We tested on the ChEMBL dataset [Mendez et al., 2019] and two curated smaller molecular datasets targeting two proteins, and found that the resulting generated molecules are much more similar to the training set as measured by Wasserstein distance of specific chemical properties and activity rate towards the protein targets.

In section 5, we study diffusion models for molecule generation. Particularly, how we can incorporate chemistry methods to improve the stability of the output molecules, we select the chemistry package xTB with the GFN2-xTB method [Bannwarth et al., 2019] to conduct quantum chemistry calculation of atom forces in molecules. We treat the chemistry package as a non-differentiable oracle and use a two-point method to estimate the gradient of the data point in the chemical property value landscape. We extend the guidance for denoising diffusion models using a differentiable regressor or classifier to non-differentiable oracles by applying guidance with the estimated gradient during the sampling process. When testing on the QM9 [Ramakrishnan et al., 2014] dataset and GEOM dataset [Axelrod and Gómez-Bombarelli], the resulting outputs have a

lower average net force on each atom and a higher percentage of stable molecules than without applying xTB guidance.

Throughout this thesis, our approaches to improving generation quality and achieving certain objectives among the generated samples can be divided into two ways – improving the underlying generative models' ability to recover the original data distribution (Chapter 3 and 4) and incorporating domain knowledge of the discrete structured data in the training and sampling process (Chapter 2 and 5). In Chapter 3 and 4, our objective is to generate data similar to the training in certain criteria – for synthetic data, our goal was to generate data with similar manifold and density as the ground-truth data and for molecular data, we want to generate data with similar property distribution as the ground-truth data. Here, the property distribution can be viewed as a proxy for the manifold and density of the original training molecular data. By introducing a "better" generative model to train on a sufficient amount of data reflective of the desired properties, the model already yields results that roughly satisfy the objectives. In Chapter 2 and 5, the problem settings are different in that the generative model or the training data is insufficient to replicate the desired properties in the generated data, thus we introduce hard constraints based on domain knowledge to reduce the degree of freedom during the training or sampling process in order to guide the generation process to fulfill the objective. Overall, this thesis leverages soft constraints (training data distribution) and hard constraints (domain-specific knowledge) to achieve generation objectives.

Here, we also would like to draw parallel between Chapter 2 and Chapter 5. An interesting follow-up on applying non-differentiable guidance during diffusion sampling is to use RL fine-tuning for the sampling process as proposed by Clark et al. [2023]. This method involves fine-tuning additional parameters to the diffusion model using a reward function calculated based on the final sampling results and how well they satisfy the desired objective. The advantage of this method is removing the need to estimate the clean data point at every step in order to calculate the gradient for guidance, as the reward is calculated based on the final diffusion sampling result. The gradient updates to the additional parameters serve to guide the sampling process similar to the diffusion guidance methods. In the case where the reward function is non-differentiable, its solution also relates to the program synthesis problem via non-differentiable renderer in Chapter 2.

Discussion Fully recovering the original data distribution is an immensely difficult task to achieve. Particularly, the evaluation for manifold and density recovery of high-dimensional real-life data, such as image and language, is difficult to conduct as the underlying distribution is unknown. The commonly used evaluation metrics such as BLEU score [Papineni et al., 2002] and FID [Heusel et al., 2017] can only approximate the similarity between the true distribution and the generated distribution. Thus, it is necessary to conduct experiments on synthetic data with a known distribution, such as the ones in Chapter 3 where we found that VAE is not guaranteed to recover nonlinear data's manifold. Pidstrigach [2022] conducted a similar study on a score-based generative model (SGM) and outlined the theoretical conditions for the full support of data distribution to be recovered. However, the density is not guaranteed to be recovered by SGM either.

Under the limitations of current generative models, integrating domain knowledge into the

generation process is a great way to induce outputs with the intended properties and features. For example, for the program synthesis project, simply keeping track of a grammar stack during generation and filtering out the invalid options according to CFG in each iteration can reduce the search space exponentially, making a problem that would need a large amount of computing resources to achieve achievable on one GPU node. Without encoding domain knowledge during the generation process, we would often require more data and computational resources. In the research space of drug discovery and molecule generation, available molecules for training are generally not as abundant as language and image data. Incorporating domain knowledge is an essential way to improve generation quality when bringing in more data is not an option. For example, in DiffDock [Corso et al., 2022], limiting the degree of freedom of the relevant molecule poses allows the diffusion model to generate state-of-the-art ligand poses for protein docking, significantly improving upon the previous approaches. This is the most successful deep learning approach, as other regression-based deep learning approaches do not perform as well as the traditional search-based approaches. In Chapter 5, we incorporate a chemistry package that calculates net forces on each atom to enable guided sampling that leads to more stable molecular configurations. This is another way to infuse domain knowledge into a machine learning system to improve generation results. However, the downside of this approach is the computational cost of accurately calculating the desired properties using quantum mechanical methods. Though the package xTB we used can infer an output within a second or so, the speed does come with a lower level of accuracy while the more accurate methods can take hours to calculate, especially with larger molecules.

Looking ahead, as people's understanding of machine learning models deepens and their interests in scientific applications grow, I believe we will see an increasing number of approaches that combine machine learning models with domain knowledge guidance in order to achieve objectives constrained by factors such as data or computational resources. Machine learning has seen fast development in model and computational capabilities. Similar developments in scientific computations, whether in computational speed or accuracy, are critical for the scientific communities to reap the benefits of the maturing machine learning capabilities. The collaboration between a scientific community and a machine learning community, like any other interdisciplinary research, requires a mutual understanding of each other's domain knowledge. This could be challenging as researchers in each field have very different educational backgrounds. A lack of understanding in the machine learning techniques or the underlying scientific problems could all produce misguided research. Despite these challenges, many successful collaborations are happening across industry and academia and I am positive that machine learning will continue to revolutionize scientific discovery in the coming years.

Bibliography

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 1.1

David Alvarez-Melis and Tommi S Jaakkola. Tree-structured decoding with doubly-recurrent neural networks. 2016. 2.2.4

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. 2

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017. 1.1, 2

Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. *Advances in Neural Information Processing Systems*, 32:7413–7424, 2019. 3.1.1

Simon Axelrod and Rafael Gómez-Bombarelli. GEOM, energy-annotated molecular conformations for property prediction and molecular generation. 9(1):185. ISSN 2052-4463. doi: 10.1038/s41597-022-01288-4. URL `https://www.nature.com/articles/s41597-022-01288-4`. 5.5, 6

Christoph Bannwarth, Eike Caldeweyher, Sebastian Ehlert, Andreas Hansen, Philipp Pracht, Jakob Seibert, Sebastian Spicher, and Stefan Grimme. Extended tight-binding quantum chemistry methods. 11(2):e1493. ISSN 1759-0876, 1759-0884. doi: 10.1002/wcms.1493. URL `https://wires.onlinelibrary.wiley.com/doi/10.1002/wcms.1493`. 5.3

Christoph Bannwarth, Sebastian Ehlert, and Stefan Grimme. Gfn2-xtb—an accurate and broadly parametrized self-consistent tight-binding quantum chemical method with multipole electrostatics and density-dependent dispersion contributions. *Journal of chemical theory and computation*, 15(3):1652–1671, 2019. 5.1, 5.3, 5.4.2, 6

Arpit Bansal, Hong-Min Chu, Avi Schwarzschild, Soumyadip Sengupta, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Universal guidance for diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 843–852, 2023. 5.1, 5.2

Fan Bao, Min Zhao, Zhongkai Hao, Peiyao Li, Chongxuan Li, and Jun Zhu. Equivariant energy-guided sde for inverse molecular design. In *The eleventh international conference on learning representations*, 2022. 5.1, 5.2, 5.4, 5.5.2

Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 1–6, 2018. 2.1

Guy W Bemis and Mark A Murcko. The properties of known drugs. 1. molecular frameworks. *Journal of medicinal chemistry*, 39(15):2887–2893, 1996. 4.4.1

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8): 1798–1828, 2013. 1

G Richard Bickerton, Gaia V Paolini, Jérémy Besnard, Sorel Muresan, and Andrew L Hopkins. Quantifying the chemical beauty of drugs. *Nature chemistry*, 4(2):90–98, 2012. 4.1, 4.4.1

Thomas Blaschke, Marcus Olivecrona, Ola Engkvist, Jürgen Bajorath, and Hongming Chen. Application of generative autoencoder in de novo molecular design. *Molecular informatics*, 37 (1-2):1700123, 2018. 4.1

Danail Bonchev. *Chemical graph theory: introduction and fundamentals*, volume 1. CRC Press, 1991. 4.1

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018. 1.1

Nathan Brown, Marco Fiscato, Marwin HS Segler, and Alain C Vaucher. Guacamol: benchmarking models for de novo molecular design. *Journal of chemical information and modeling*, 59 (3):1096–1108, 2019. 4.1

Rudy Bunel, Matthew Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. *arXiv preprint arXiv:1805.04276*, 2018. 2, 2.3.1

Yudong Cao, Jonathan Romero, Jonathan P. Olson, Matthias Degroote, Peter D. Johnson, Mária Kieferová, Ian D. Kivlichan, Tim Menke, Borja Peropadre, Nicolas P. D. Sawaya, Sukin Sim, Libor Veis, and Alán Aspuru-Guzik. Quantum chemistry in the age of quantum computing. 119(19):10856–10915. ISSN 0009-2665, 1520-6890. doi: 10.1021/acs.chemrev.8b00803. URL `https://pubs.acs.org/doi/10.1021/acs.chemrev.8b00803`. 5.3

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, 2016. 2

Kevin Clark, Paul Vicol, Kevin Swersky, and David J Fleet. Directly fine-tuning diffusion models on differentiable rewards. *arXiv preprint arXiv:2309.17400*, 2023. 6

Gabriele Corso, Hannes Stärk, Bowen Jing, Regina Barzilay, and Tommi Jaakkola. Diffdock: Diffusion steps, twists, and turns for molecular docking. *arXiv preprint arXiv:2210.01776*, 2022. 1, 5.1, 6

Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012. 2.2.2

Bin Dai and David Wipf. Diagnosing and enhancing vae models. *arXiv preprint arXiv:1903.05789*, 2019. 3.1, 3.1.1, 3.3, 3.3, 3.3, 3.3, 3.3, 3.4, 3.4.1, 3.4, 4.3, 4.3.2, 4.3.2, 4.3.2, 4.4.1, 4.4.1

Bin Dai, Yu Wang, John Aston, Gang Hua, and David Wipf. Hidden talents of the variational autoencoder. *arXiv preprint arXiv:1706.05148*, 2017. 3.1, 3.1.1

Bin Dai, Ziyu Wang, and David Wipf. The usual suspects? reassessing blame for vae posterior collapse. In *International Conference on Machine Learning*, pages 2313–2322. PMLR, 2020. 3.1.1

Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. Syntax-directed variational autoencoder for structured data. *arXiv preprint arXiv:1802.08786*, 2018. 4.2, 4.3

Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018. 5.2

Jörg Degen, Christof Wegscheid-Gerlach, Andrea Zaliani, and Matthias Rarey. On the art of compiling and using'drug-like'chemical fragment spaces. *ChemMedChem: Chemistry Enabling Drug Discovery*, 3(10):1503–1507, 2008. 4.4.1

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 1

Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis, 2021. 1.1, 5.1, 5.2, 5.3, 5.4.1

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 1.1

Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. Inversecsg: Automatic conversion of 3d models to csg trees. *ACM Transactions on Graphics (TOG)*, 37(6):1–16, 2018. 2.1

David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. *arXiv preprint arXiv:1509.09292*, 2015. 4.1, 4.2

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to infer graphics programs from hand-drawn images. In *Advances in neural information processing systems*, pages 6059–6068, 2018. 2.1

Kevin Ellis, Maxwell Nye, Yewen Pu, Felix Sosa, Josh Tenenbaum, and Armando Solar-Lezama. Write, execute, assess: Program synthesis with a repl, 2019. 2, 2.1

Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of cheminformatics*, 1(1):1–11, 2009. 4.1, 4.4.1

Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049, 2016. 4.3

Hao Fu, Chunyuan Li, Xiaodong Liu, Jianfeng Gao, Asli Celikyilmaz, and Lawrence Carin. Cyclical annealing schedule: A simple approach to mitigating kl vanishing. *arXiv preprint arXiv:1903.10145*, 2019. 4.4.1

Johannes Gasteiger, Shankari Giri, Johannes T. Margraf, and Stephan Günnemann. Fast and uncertainty-aware directional message passing for non-equilibrium molecules. In *Machine Learning for Molecules Workshop, NeurIPS*, 2020. 5.4.2

Niklas Gebauer, Michael Gastegger, and Kristof Schütt. Symmetry-adapted generation of 3d point sets for the targeted discovery of molecules. *Advances in neural information processing systems*, 32, 2019. 4.2, 5.2

Niklas WA Gebauer, Michael Gastegger, Stefaan SP Hessmann, Klaus-Robert Müller, and Kristof T Schütt. Inverse design of 3d molecular structures with conditional generative neural networks. *Nature communications*, 13(1):1–11, 2022. 4.2

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018. 4.1, 4.2

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 1, 1.1, 2

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016. 1, 3.1, 4.3

Alex Graves and Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pages 37–45, 2012. 1.1

Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Carlos Outeiral, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. Objective-reinforced generative adversarial networks (organ) for sequence generation models. *arXiv preprint arXiv:1705.10843*, 2017. 4.2

Suriya Gunasekar, Blake Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nathan Srebro. Implicit regularization in matrix factorization. In *2018 Information Theory and Applications Workshop (ITA)*, pages 1–10. IEEE, 2018. 3.1.1

Xu Han, Caihua Shan, Yifei Shen, Can Xu, Han Yang, Xiang Li, and Dongsheng Li. Training-free multi-objective diffusion model for 3d molecule generation. In *The Twelfth International Conference on Learning Representations*, 2023. 5.1, 5.2, 5.4, 5.5.2

Xu Han, Caihua Shan, Yifei Shen, Can Xu, Han Yang, Xiang Li, and Dongsheng Li. Training-free multi-objective diffusion model for 3d molecule generation. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=X41c4uB4k0`. (document), 5.2, 5.5.1

Junxian He, Daniel Spokoyny, Graham Neubig, and Taylor Berg-Kirkpatrick. Lagging inference networks and posterior collapse in variational autoencoders. *arXiv preprint arXiv:1901.05534*, 2019. 3.1.1

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 6

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick,

Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR*, 2017. 2

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 1, 1.1, 1.1, 5.3, 5.3

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997. 1.1

Emiel Hoogeboom, Victor Garcia Satorras, Clement Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. In *International Conference on Machine Learning*, pages 8867–8887. PMLR, 2022. 4.1, 4.2, 5.1, 5.2, 5.5

Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012. 3.5

Philip M Hubbard. *Constructive solid geometry for triangulated polyhedra*. 1990. 2, 2.2

John J Irwin and Brian K Shoichet. Zinc- a free database of commercially available compounds for virtual screening. *Journal of chemical information and modeling*, 45(1):177–182, 2005. 4.1

Arthur Jacot, François Ged, Franck Gabriel, Berfin Şimşek, and Clément Hongler. Deep linear networks dynamics: Low-rank biases induced by initialization scale and l2 regularization. *arXiv preprint arXiv:2106.15933*, 2021. 3.1.1

Ziwei Ji and Matus Telgarsky. Gradient descent aligns the layers of deep linear networks. *arXiv preprint arXiv:1810.02032*, 2018. 3.1.1

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning*, pages 2328–2337, 2018. 4.1, 4.2, 5.2

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical graph-to-graph translation for molecules. *arXiv preprint arXiv:1907.11223*, 2019. 4.3, 5.2

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical generation of molecular graphs using structural motifs. In *International Conference on Machine Learning*, pages 4839–4848. PMLR, 2020a. 4.1, 4.2, 4.4.1, 4.4.3

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Multi-objective molecule generation using interpretable substructures. In *International Conference on Machine Learning*, pages 4849–4859. PMLR, 2020b. 4.1, 4.4, 4.4.4, 4.4.4, 4.4.4, 4.5

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021. 1, 5.1

Artur Kadurin, Sergey Nikolenko, Kuzma Khrabrov, Alex Aliper, and Alex Zhavoronkov. drugan: an advanced generative adversarial autoencoder model for de novo generation of new molecules with desired molecular properties in silico. *Molecular pharmaceutics*, 14(9):3098–3104, 2017. 4.2

Kacper Kania, Maciej Zięba, and Tomasz Kajdanowicz. Ucsg-net–unsupervised discovering of constructive solid geometry tree. *arXiv preprint arXiv:2006.09102*, 2020. 2, 2.1

Bahjat Kawar, Michael Elad, Stefano Ermon, and Jiaming Song. Denoising diffusion restoration

models. *Advances in Neural Information Processing Systems*, 35:23593–23606, 2022. 5.4.1

Hyunjik Kim and Andriy Mnih. Disentangling by factorising. *arXiv preprint arXiv:1802.05983*, 2018. 2

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 1, 1.1, 2, 4.3

Frederic Koehler, Viraj Mehta, Andrej Risteski, and Chenghui Zhou. Variational autoencoders in the presence of low-dimensional data: landscape and implicit bias. *arXiv preprint arXiv:2112.06868*, 2021. 1, 4.3, 4.3.2, 4.3.2, 4.3.2

Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 reinforce samples, get a baseline for free! 2019a. 2.2.3

Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, pages 3499–3508, 2019b. 6, 2.2.2, 2.2.3, 2.2.3

Maria Korshunova, Niles Huang, Stephen Capuzzi, Dmytro S Radchenko, Olena Savych, Yuriy S Moroz, Carrow I Wells, Timothy M Willson, Alexander Tropsha, and Olexandr Isayev. Generative and reinforcement learning approaches for the automated de novo design of bioactive compounds. *Communications Chemistry*, 5(1):129, 2022. 4.4.4

Abhishek Kumar and Ben Poole. On implicit regularization in β-vaes. In *International Conference on Machine Learning*, pages 5480–5490. PMLR, 2020. 3.1.1

Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1945–1954. JMLR. org, 2017. 2.2.4, 4.2, 4.3, 5.2

Adam Daniel Laud. Theory and application of reward shaping in reinforcement learning. Technical report, 2004. 2.2.1

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 3.8

Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. Mmd gan: Towards deeper understanding of moment matching network. In *Advances in Neural Information Processing Systems*, pages 2203–2213, 2017. 1.1, 2

Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. In *SIGGRAPH Asia 2018 Technical Papers*, page 222. ACM, 2018a. 2

Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. In *Conference On Learning Theory*, pages 2–47. PMLR, 2018b. 3.1.1

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. 1

Zhiyuan Li, Yuping Luo, and Kaifeng Lyu. Towards resolving the implicit bias of gradient descent

for matrix factorization: Greedy low-rank learning. *arXiv preprint arXiv:2012.09839*, 2020. 3.1.1

Hsueh-Ti Derek Liu, Michael Tao, Chun-Liang Li, Derek Nowrouzezahrai, and Alec Jacobson. Beyond pixel norm-balls: Parametric adversaries using an analytically differentiable renderer. 2019. 2

Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L Gaunt. Constrained graph variational autoencoders for molecule design. *arXiv preprint arXiv:1805.09076*, 2018a. 4.1, 4.2

Yunchao Liu, Zheng Wu, Daniel Ritchie, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Learning to describe scenes with programs. 2018b. 2.1

Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Rätsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. *arXiv preprint arXiv:1811.12359*, 2018. 2

James Lucas, George Tucker, Roger B Grosse, and Mohammad Norouzi. Don't blame the elbo! a linear vae perspective on posterior collapse. *Advances in Neural Information Processing Systems*, 32:9408–9418, 2019. 3.1.1, 3.3, 3.6

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11461–11471, 2022. 1

Shitong Luo, Jiaqi Guan, Jianzhu Ma, and Jian Peng. A 3d generative model for structure-based drug design. *Advances in Neural Information Processing Systems*, 34:6229–6239, 2021. 4.2

Youzhi Luo and Shuiwang Ji. An autoregressive flow model for 3d molecular geometry generation from scratch. In *International Conference on Learning Representations (ICLR)*, 2022. 5.2

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alexandru Damian, Jason D. Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. *ArXiv*, abs/2305.17333, 2023. URL `https://api.semanticscholar.org/CorpusID:258959274`. 5.4.2

Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017. 3.8

Krzysztof Maziarz, Henry Jackson-Flux, Pashmina Cameron, Finton Sirockin, Nadine Schneider, Nikolaus Stiefl, Marwin Segler, and Marc Brockschmidt. Learning to extend molecular scaffolds with structural motifs. *arXiv preprint arXiv:2103.03864*, 2021. 4.1, 4.2, 4.4, 4.4.1, 5.2

David Mendez, Anna Gaulton, A Patrícia Bento, Jon Chambers, Marleen De Veij, Eloy Félix, María Paula Magariños, Juan F Mosquera, Prudence Mutowo, Michał Nowotka, et al. Chembl: towards direct deposition of bioassay data. *Nucleic acids research*, 47(D1):D930–D940, 2019. 1, 4.1, 4.4.1, 6

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 2.2.1

Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017. 5.1, 5.4.2

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016. 1, 2

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *arXiv preprint arXiv:1711.00937*, 2017. 1, 3.1, 3.1.1

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002. 6

Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. *arXiv preprint arXiv:1611.01855*, 2016. 2.2.4

Adeel Pervez and Efstratios Gavves. Spectral smoothing unveils phase transitions in hierarchical variational autoencoders. In *International Conference on Machine Learning*, pages 8536–8545. PMLR, 2021. 3.1.1

Jakiw Pidstrigach. Score-based generative models detect manifolds. *Advances in Neural Information Processing Systems*, 35:35852–35865, 2022. 6

Daniil Polykovskiy, Alexander Zhebrak, Benjamin Sanchez-Lengeling, Sergey Golovanov, Oktai Tatanov, Stanislav Belyaev, Rauf Kurbanov, Aleksey Artamonov, Vladimir Aladinskiy, Mark Veselov, et al. Molecular sets (moses): a benchmarking platform for molecular generation models. *Frontiers in pharmacology*, 11:1931, 2020. 4.1, 4.3.2, 4.4.1

Kristina Preuer, Philipp Renz, Thomas Unterthiner, Sepp Hochreiter, and Gunter Klambauer. Fréchet chemnet distance: a metric for generative models for molecules in drug discovery. *Journal of chemical information and modeling*, 58(9):1736–1741, 2018. 4.4.1

Oleksii Prykhodko, Simon Viet Johansson, Panagiotis-Christos Kotsias, Josep Arús-Pous, Esben Jannik Bjerrum, Ola Engkvist, and Hongming Chen. A de novo molecular generation method using latent vector based generative adversarial network. *Journal of Cheminformatics*, 11(1):1–13, 2019. 4.2, 4.4.1, 5.2

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 1

Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014. 4.2, 5.1, 5.5, 6

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International conference on machine learning*, pages 8821–8831. Pmlr, 2021. 1

Ali Razavi, Aäron van den Oord, Ben Poole, and Oriol Vinyals. Preventing posterior collapse with delta-vaes. *arXiv preprint arXiv:1901.03416*, 2019a. 3.1.1, 4.4.1

Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019b. 1.1, 3.1, 3.1.1

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015. 1

David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 50(5):742–754, 2010. 4.4.1

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 1, 1.1, 5.2, 5.3

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023. 1

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. 1.1

Victor Garcia Satorras, Emiel Hoogeboom, Fabian B Fuchs, Ingmar Posner, and Max Welling. E (n) equivariant normalizing flows. *arXiv preprint arXiv:2105.09016*, 2021. 4.1, 5.3

Robert E Schapire and Yoav Freund. Boosting: Foundations and algorithms. *Kybernetes*, 2013. 3.1.1

Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science*, 4 (1):120–131, 2018. 4.1, 4.2

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Csgnet: Neural shape parser for constructive solid geometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5515–5523, 2018. 2, 2.1, 2.2, 2.2.1, 2.2.1, 2.2.4, 2.3.2

Richard Shin, Neel Kant, Kavi Gupta, Christopher Bender, Brandon Trabucco, Rishabh Singh, and Dawn Song. Synthetic datasets for neural program synthesis. *arXiv preprint arXiv:1912.12345*, 2019. 2.1

Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International conference on artificial neural networks*, pages 412–422. Springer, 2018. 4.1

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015. 1.1, 5.3

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020a. 1, 1.1, 5.3, 5.4.1

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019. 1, 1.1, 1.1

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020b. 1, 1.1

Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *The Journal of Machine Learning Research*,

19(1):2822–2878, 2018. 3.1.1

James C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37:332–341, 1992. URL `https://api.semanticscholar.org/CorpusID:122365276`. 5.4.2

Peter C St. John, Caleb Phillips, Travis W Kemper, A Nolan Wilson, Yanfei Guan, Michael F Crowley, Mark R Nimlos, and Ross E Larsen. Message-passing neural networks for high-throughput polymer screening. *The Journal of chemical physics*, 150(23):234111, 2019. 4.4.3

Rong Tang and Yun Yang. On empirical bayes variational autoencoder: An excess risk bound. In *Conference on Learning Theory*, pages 4068–4125. PMLR, 2021. 3.1.1

Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Learning to infer and execute 3d shape programs. *arXiv preprint arXiv:1901.02875*, 2019. 2.1

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 1.1

Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *arXiv preprint arXiv:2007.03898*, 2020. 1, 1.1, 3.1, 3.1.1

Aaron van den Oord, Oriol Vinyals, et al. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pages 6306–6315, 2017. 1.1, 2

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1.1

Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. *arXiv preprint arXiv:2209.14734*, 2022. 4.2, 5.1, 5.2, 5.4, 5.4.1

David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28 (1):31–36, 1988. 4.1

Scott A Wildman and Gordon M Crippen. Prediction of physicochemical parameters by atomic contributions. *Journal of chemical information and computer sciences*, 39(5):868–873, 1999. 4.1, 4.4.1

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 2.2.1

Robin Winter, Floriane Montanari, Andreas Steffen, Hans Briem, Frank Noé, and Djork-Arné Clevert. Efficient multi-objective molecular optimization in a continuous latent space. *Chemical science*, 10(34):8016–8024, 2019. 4.1

Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. Geodiff: A geometric diffusion model for molecular conformation generation. *arXiv preprint arXiv:2203.02923*, 2022. 4.2

Minkai Xu, Alexander S Powers, Ron O Dror, Stefano Ermon, and Jure Leskovec. Geometric latent diffusion models for 3d molecule generation. In *International Conference on Machine Learning*, pages 38592–38610. PMLR, 2023. 5.1, 5.2, 5.3, 5.3, 5.5

Kevin Yang, Kyle Swanson, Wengong Jin, Connor Coley, Philipp Eiden, Hua Gao, Angel Guzman-Perez, Timothy Hopper, Brian Kelley, Miriam Mathea, et al. Analyzing learned molecular representations for property prediction. *Journal of chemical information and modeling*, 59(8): 3370–3388, 2019. 4.4.4

Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*, 2017. 2.2.4

Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *arXiv preprint arXiv:1806.02473*, 2018. 4.1

Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Generative image inpainting with contextual attention. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5505–5514, 2018. 1

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3836–3847, 2023. 5.2

Chenghui Zhou and Barnabas Poczos. Objective-agnostic enhancement of molecule properties via multi-stage vae. *arXiv preprint arXiv:2308.13066*, 2023. 1

Chenghui Zhou, Chun-Liang Li, and Barnabás Póczos. Unsupervised program synthesis for images by sampling without replacement. In *Uncertainty in Artificial Intelligence*, pages 408–418. PMLR, 2021. 1

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017. 1.1