Grammar Variational Autoencoder

Matt J. Kusner [1 2] **Brooks Paige** [1 3] **José Miguel Hernández-Lobato** [3]

Abstract

Deep generative models have been wildly successful at learning coherent latent representations for continuous data such as video and audio. However, generative modeling of discrete data such as arithmetic expressions and molecular structures still poses significant challenges. Crucially, state-of-the-art methods often produce outputs that are not valid. We make the key observation that frequently, discrete data can be represented as a parse tree from a context-free grammar. We propose a variational autoencoder which encodes and decodes directly to and from these parse trees, ensuring the generated outputs are always valid. Surprisingly, we show that not only does our model more often generate valid outputs, it also learns a more coherent latent space in which nearby points decode to similar discrete outputs. We demonstrate the effectiveness of our learned models by showing their improved performance in Bayesian optimization for symbolic regression and molecular synthesis.

1. Introduction

Generative machine learning models have been used recently to produce extraordinary results, from realistic musical improvisation (Jaques et al., 2016), to changing facial expressions in images (Radford et al., 2015; Upchurch et al., 2016), to creating realistic looking artwork (Gatys et al., 2015). In large part, these generative models have been successful at representing data in continuous domains. Recently there is increased interest in training generative models to construct more complex, discrete data types such as arithmetic expressions (Kusner & Hernández-Lobato, 2016), source code (Gaunt et al., 2016; Riedel et al., 2016) and molecules (Gómez-Bombarelli et al., 2016b).

To train generative models for these tasks, these objects are often first represented as strings. This is in large part due to the fact that there exist powerful models for text sequence modeling such as Long Short Term Memory networks (LSTMs) (Hochreiter & Schmidhuber, 1997), Gated Recurrent Units (GRUs) (Cho et al., 2014), and Dynamic Convolutional Neural Networks (DCNNs) (Kalchbrenner et al., 2014). For instance, molecules can be represented by so-called SMILES strings (Weininger, 1988) and Gómez-Bombarelli et al. (2016b) has recently developed a generative model for molecules based on SMILES strings that uses GRUs and DCNNs. This model is able to encode and decode molecules to and from a continuous latent space, allowing one to search this space for new molecules with desirable properties (Gómez-Bombarelli et al., 2016b).

However, one immediate difficulty in using strings to represent molecules is that the representation is very brittle: small changes in the string can lead to completely different molecules, or often do not correspond to valid molecules at all. Specifically, Gómez-Bombarelli et al. (2016b) described that while searching for new molecules, the probabilistic decoder — the distribution which maps from the continuous latent space into the space of molecular structures — would sometimes accidentally put high probability on strings which are not valid SMILES strings or do not encode plausible molecules.

To address this issue, we propose to directly incorporate knowledge about the structure of discrete data using a *grammar*. Grammars exist for a wide variety of discrete domains such as symbolic expressions (Allamanis et al., 2016), standard programming languages such as C (Kernighan et al., 1988), and chemical structures (James et al., 2015). For instance the set of syntactically valid SMILES strings is described using a context free grammar, which can be used for parsing and validation.

Given a grammar, every valid discrete object can be described as a parse tree from the grammar. Thus, we propose the *grammar variational autoencoder* (GVAE) which encodes and decodes directly to and from these parse trees. Generating parse trees as opposed to text ensures that all outputs are valid based on the grammar. This frees the GVAE from learning syntactic rules and allows it to wholly focus on learning other 'semantic' properties.

We demonstrate the GVAE on two different tasks for gen-

[1]Alan Turing Institute [2]University of Warwick [3]University of Cambridge. Correspondence to: <mkusner@turing.ac.uk>, <bpaige@turing.ac.uk>, <jmh233@cam.ac.uk>.

http://opensmiles.org/spec/open-smiles-2-grammar.html

erating discrete data: 1) generating simple arithmetic expressions and 2) generating valid molecules. We show not only does our model produce a higher proportion of valid discrete outputs than a character based autoencoder, it also produces smoother latent representations. We also show that this learned latent space is effective for searching for arithmetic expressions that fit data, for finding better drug-like molecules, and for making accurate predictions about target properties.

2. Background

2.1. Variational autoencoder

We wish to learn both an encoder and a decoder for mapping data $\mathbf{x}$ to and from values $\mathbf{z}$ in a continuous space. The variational autoencoder (Kingma & Welling, 2014; Rezende et al., 2014) provides a formulation in which the encoding $\mathbf{z}$ is interpreted as a latent variable in a probabilistic generative model; a probabilistic decoder is defined by a likelihood function $p_\theta(\mathbf{x}|\mathbf{z})$ and parameterized by θ. Alongside a prior distribution $p(\mathbf{z})$ over the latent variables, the posterior distribution $p_\theta(\mathbf{z}|\mathbf{x}) \propto p(\mathbf{z})p_\theta(\mathbf{x}|\mathbf{z})$ can then be interpreted as a probabilistic encoder.

To admit efficient inference, the variational Bayes approach simultaneously learns both the parameters of $p_\theta(\mathbf{x}|\mathbf{z})$ as well as those of a posterior approximation $q_\phi(\mathbf{z}|\mathbf{x})$. This is achieved by maximizing the evidence lower bound (ELBO)

$$\mathcal{L}(\phi, \theta; \mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}) \right], \quad (1)$$

with $\mathcal{L}(\phi, \theta; \mathbf{x}) \leq \log p_\theta(\mathbf{x})$. So long as $p_\theta(\mathbf{x}|\mathbf{z})$ and $q_\phi(\mathbf{z}|\mathbf{x})$ can be computed pointwise, and are differentiable with respect to their parameters, the ELBO can be maximized via gradient descent; this allows wide flexibility in choice of encoder and decoder models. Typically these will take the form of exponential family distributions whose parameters are the output of a multi-layer neural network.

2.2. Context-free grammars

A context-free grammar (CFG) is traditionally defined as a 4-tuple $G = (V, \Sigma, R, S)$: V is a finite set of non-terminal symbols; the *alphabet* Σ is a finite set of terminal symbols, disjoint from V; R is a finite set of production rules; and S is a distinct non-terminal known as the *start symbol*. The rules R are formally described as $\alpha \to \beta$ for $\alpha \in V$ and $\beta \in (V \cup \Sigma)^*$, with * denoting the Kleene closure. In practice, these rules are defined as a set of mappings from a single left-hand side non-terminal in V to a sequence of terminal and/or non-terminal symbols, and can be interpreted as a rewrite rule.

Application of a production rule to a non-terminal symbol defines a tree, with symbols on the right-hand side of the production rule becoming child nodes for the left-hand side parent. The grammar G thus defines a set of possible trees extending from each non-terminal symbol in V, produced by recursively applying rules in R to leaf nodes until all leaf nodes are terminal symbols in Σ. The *language* of G is set of all sequences of terminal symbols which can be produced by a left-to-right traversal of the leaf nodes in a tree. Given a string in the language (i.e., a sequence of terminals), a *parse tree* is a tree rooted at S which has this sequence of terminal symbols as its leaf nodes. The ubiquity of context-free languages in computer science is due in part to the presence of efficient parsing algorithms. For more background on context free grammars and automata theory, see e.g. Hopcroft et al. (2006).

The context-free grammar can form the backbone of a probabilistic generative model for valid strings. By assigning probabilities to each production rule in the grammar, it is possible to define a probability distribution over parse trees (Baker, 1979; Booth & Thompson, 1973). A string can be generated by repeatedly sampling and applying production rules, beginning from the start symbol, until no non-terminals remain. Modern approaches allow the probabilities used to at each stage to depend on the current state of the parse tree (Johnson et al., 2007).

3. Methods

In this section we describe how a grammar can improve variational autoencoders (VAE) for discrete data. It will do so by drastically reducing the number of invalid outputs generated from the VAE.

One glaring issue with the character VAE is that it may frequently map latent points to sequences that are not valid, hoping the VAE will infer from training data what constitutes a valid sequence. Instead of implicitly encouraging the VAE to produce valid molecules, we propose to give the VAE explicit knowledge about how to produce valid molecules. We do this by using a grammar for the sequences: given a grammar we can take any valid sequence and parse it into a sequence of production rules. Applying these rules in order will yield the original sequence. Our approach will be to learn a VAE that produces sequences of grammar production rules. The benefit is that it is trivial to generate valid sequences of production rules, as the grammar describes the valid set of rules that can be selected at any point during the generation process. Thus our model is able to focus on learning semantic properties of sequence data without also having to learn syntactic constraints. We describe our model in detail on a small example.

3.1. An illustrative example

We propose a grammar variational autoencoder (GVAE) that encodes and decodes in the space of grammar produc-

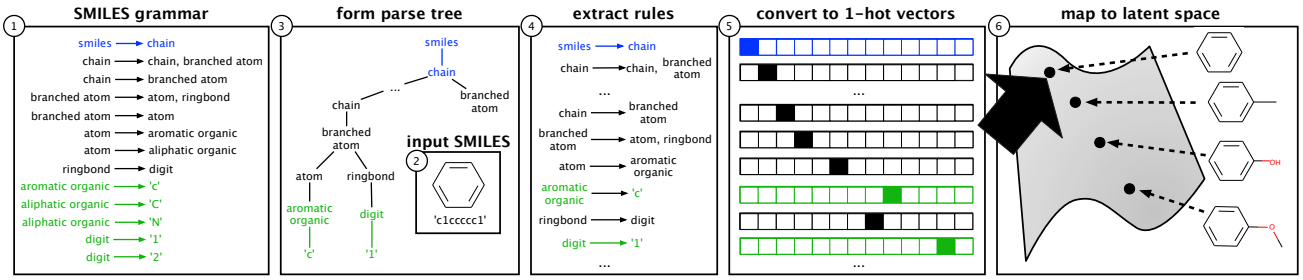

Figure 1. The encoder of the GVAE. We denote the start rule in blue and all rules that decode to terminal in green. See text for details.

tion rules. We describe how the GVAE works using a simple example.

Encoding. Consider a subset of the SMILES grammar as shown in Figure 1, box (1). These are the possible production rules that can be used for constructing a molecule. Imagine we are given as input the SMILES string for benzene: 'c1ccccc1'. Figure 1, box (2) shows this molecule. To encode this molecule into a continuous latent representation we begin by using the SMILES grammar to parse this string into a parse tree (partially shown in box (3)). This tree describes how 'c1ccccc1' is generated by the grammar. We decompose this tree into a sequence of production rules by performing a pre-order traversal on the branches of the parse tree going from left-to-right, shown in box (4). We convert these rules into 1-hot indicator vectors, where each dimension corresponds to a rule in the SMILES grammar, box (5). Letting K denote the total number of production rules in the entire grammar, and $T(\mathbf{X})$ the number of productions applied in total to generate the output string for $\mathbf{X}$, the collection of 1-hot vectors can be written as a $T(\mathbf{X}) \times K$ matrix $\mathbf{X}$. We use a deep convolutional neural network to map this collection of 1-hot vectors $\mathbf{X}$ to a continuous latent vector $\mathbf{z}$ The architecture of the encoding network is described in the supplementary material.

Decoding. We now describe how we map continuous vectors back to a sequence of production rules (and thus SMILES strings). Crucially we construct the decoder so that at any time while we are decoding this sequence the decoder will only be allowed to select a subset of production rules that are 'valid'. This will cause the decoder to only produce valid parse sequences from the grammar.

We begin by passing the continuous vector $\mathbf{z}$ through a recurrent neural network which produces a set of unnormalized log probability vectors (or 'logits'), shown in Figure 2, box (1) and (2). Exactly like the 1-hot vectors produced by the encoder, each dimension of the logit vectors corresponds to a production rule in the grammar. We can again write these collection of logit vectors as a matrix $\mathbf{F} \in \mathbb{R}^{T_{max} \times K}$, where T_{max} is the maximum number of timesteps (production rules) allowed by the decoder. We will use these vectors in the rest of the decoder to select production rules.

To ensure that any sequence of production rules generated from the decoder is valid, we keep track of the state of the parsing using a last-in first-out (LIFO) stack. This is shown in Figure 2, box (3). At the beginning, every valid parse from the grammar must start with the start symbol: *smiles*, which is placed on the stack. Next we pop off whatever non-terminal symbol that was placed last on the stack (in this case *smiles*), and we use it to mask out the invalid dimensions of the logit vector. Formally, for every non-terminal α we define a fixed binary mask vector $\mathbf{m}_\alpha \in [0, 1]^K$. This takes the value '1' for all indices in $1, \ldots, K$ corresponding to production rules that have α on their left-hand-side.

In this case the only production rule in the grammar beginning with *smiles* is the first so we zero-out every dimension except the first, shown in Figure 2, box (4). We then sample from the remaining unmasked rules, using their values in the logit vector. To sample from this masked logit at any timestep t we form the following masked distribution:

$$p(\mathbf{x}_t = k|\alpha, \mathbf{z}) = \frac{m_{\alpha,k} \exp(f_{tk})}{\sum_{j=1}^{K} m_{\alpha,k} \exp(f_{tj})}, \qquad (2)$$

where f_{tk} is the (t, k)-element of the logit matrix $\mathbf{F}$. As only the first rule is unmasked we will select this rule *smiles* $\rightarrow$ *chain* as the first rule in our generated sequence.

Now the next rule must begin with *chain*, so we push it onto the stack (Figure 2, box (3)). We sample this non-terminal and again use it to mask out all of the rules that cannot be applied in the current logit vector. We then sample a valid rule from this logit vector: *chain* $\rightarrow$ *chain*, *branched atom*. Just as before we push the non-terminals on the right-hand side of this rule onto the stack, adding the individual non-terminals in from right to left, such that the leftmost non-terminal is on the top of the stack. For the next state we again pop the last rule placed on the stack and mask the current logit, etc. This process continues until the stack is empty or we reach the maximum number of logit vec-

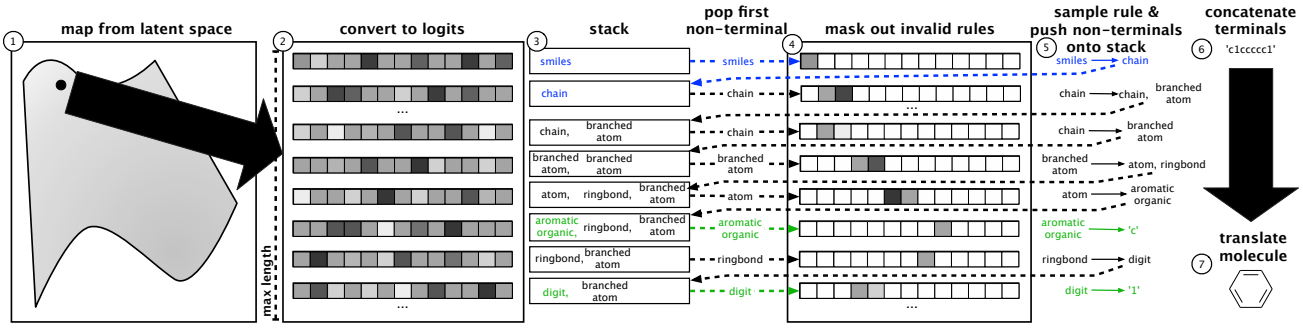

Figure 2. The decoder of the GVAE. See text for details.

Algorithm 1 Sampling from the decoder

Input: Deterministic decoder output $\mathbf{F} \in \mathbb{R}^{T_{max} \times K}$, masks $\mathbf{m}_\alpha$ for each production rule α
Output: Sampled productions $\mathbf{X}$ from $p(\mathbf{X}|\mathbf{z})$

Initialize empty stack $\mathcal{S}$, and push the start symbol S onto the top; set $t = 0$
while $\mathcal{S}$ is nonempty **do**
  Pop the last-pushed non-terminal α from the stack $\mathcal{S}$
  Use Eq. (2) to sample a production rule $\mathcal{R}$
  Set $\mathbf{x}_t \leftarrow \mathcal{R}$
  Let $\text{RHS}(\mathcal{R})$ denote all non-terminals on the right-hand side of rule $\mathcal{R}$, ordered from right to left
  for non-terminal β in $\text{RHS}(\mathcal{R})$ **do**
    Push β on to the stack $\mathcal{S}$
  end for
  Set $\mathbf{X} \leftarrow [\mathbf{X}, \mathbf{x}_t]$
  Set $t \leftarrow t + 1$
end while

tors T_{max}. We describe this decoding procedure formally in Algorithm 1. In practice, because sampling from the decoder often finishes before t reaches T_{max}, we introduce an additional 'no-op' rule to the grammar that we use to pad $\mathbf{X}$ until the number of rows equals T_{max}.

We note the explicit connection between the process in Algorithm 1 and parsing algorithms for pushdown automata. A pushdown automaton is a finite state machine which has access to a single stack for long-term storage, and are equivalent to context-free grammars in the sense that every CFG can be converted into a pushdown automaton, and vice-versa (Hopcroft et al., 2006). The decoding algorithm performs the sequence of actions taken by a nondeterministic pushdown automaton at each stage of a parsing algorithm; the nondeterminism is resolved by sampling according to the probabilities in the emitted logit vector.

Contrasting the character VAE. Notice that the key difference between this grammar VAE decoder and a character-based VAE decoder is that at every point in the generated sequence, the character VAE can sample any possible character. There is no stack or masking operation. The grammar VAE however is constrained to select syntactically-valid sequences.

Syntactic vs. semantic validity. It is important to note that the grammar encodes *syntactically valid* molecules but not necessarily *semantically valid* molecules. This is because: 1. certain molecules produced by the grammar may be very unstable molecules or not chemically-valid (for instance an oxygen atom cannot bond to 3 other atoms as it only has 2 free electrons for bonding, although it would be possible to generate this in a molecule from the grammar). 2. The SMILES language has non-context free aspects such as a ringbond must be opened and closed by the same digit, starting with '1' (such is the case for benzene 'c1ccccc1'). The particular challenge for matching digits, in contrast to matching grouping symbols such as parentheses, is that they do not compose in a nested manner; for example, 'C12(CCCCC1)CCCCC2' is a valid SMILES string and molecule. Furthermore, any intermediate ringbond must use digits that increment by one for each new ringbond. Keeping track of which digit to use for each ringbond is not context-free. 3. Finally, we note that the GVAE can output an undetermined sequence if there are still non-terminal symbols on the stack after processing all T_{max} logit vectors. While this could be fixed by a procedure that converts these non-terminals to terminals, for simplicity we mark these sequences as invalid.

3.2. Training

During training, each input SMILES encoded as a sequence of 1-hot vectors $\mathbf{X} \in \{0, 1\}^{T_{max} \times K}$, also defines a sequence of T_{max} mask vectors. Each mask at timestep $t = 1, \ldots, T_{max}$ is selected by the left-hand side of the production rule indicated in the 1-hot vector $\mathbf{x}_t$. Given these masks we can compute the decoder's mapping

$$p(\mathbf{X}|\mathbf{z}) = \prod_{t=1}^{T(\mathbf{X})} p(\mathbf{x}_t|\mathbf{z}), \tag{3}$$

Algorithm 2 Training the Grammar VAE

Input: Dataset $\{\mathbf{X}^{(i)}\}_{i=1}^{N}$
Output: Trained VAE model $p_\theta(\mathbf{X}|\mathbf{z}), q_\phi(\mathbf{z}|\mathbf{X})$

while VAE not converged **do**
  Select element: $\mathbf{X} \in \{\mathbf{X}^{(i)}\}_{i=1}^{N}$ (or minibatch)
  Encode: $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{X})$
  Decode: given $\mathbf{z}$, compute logits $\mathbf{F} \in \mathbb{R}^{T_{max} \times K}$
  for t in $[1, \ldots, T_{max}]$ **do**
    Compute $p_\theta(\mathbf{x}_t|\mathbf{z})$ via Eq. (2), with mask $\mathbf{m}_{\mathbf{x}_t}$ and logits $\mathbf{f}_t$
  end for
  Update θ, ϕ using estimates $p_\theta(\mathbf{X}|\mathbf{z}), q_\phi(\mathbf{z}|\mathbf{X})$, via gradient descent on the ELBO in Eq. (4)
end while

with the individual probabilities at each timestep defined as in Eq. (2). We pad any remaining timesteps after $T(\mathbf{X})$ up to T_{max} with a dummy rule, a one-hot vector indicating the parse tree is complete and no actions are to be taken.

In all our experiments, $q(\mathbf{z}|\mathbf{X})$ is a Gaussian distribution whose mean and variance parameters are the output of the encoder network, with an isotropic Gaussian prior $p(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$. At training time, we sample a value of $\mathbf{z}$ from $q(\mathbf{z}|\mathbf{X})$ to compute the ELBO

$$\mathcal{L}(\phi, \theta; \mathbf{X}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{X})} \left[\log p_\theta(\mathbf{X}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{X}) \right]. \quad (4)$$

Following Kingma & Welling (2014), we apply a non-centered parameterization on the encoding Gaussian distribution and optimize Eq. (4) using gradient descent, learning encoder and decoder neural network parameters ϕ and θ. Algorithm 2 summarizes the training procedure.

4. Experiments

We show the usefulness of our proposed grammar variational autoencoder (GVAE) on two sequence optimization problems: 1) searching for an arithmetic expression that best fits a dataset and 2) finding new drug molecules. We begin by showing the latent space of the GVAE and a character variational autoencoder (CVAE), similar to that of Gómez-Bombarelli et al. (2016b), on each of the problems. We demonstrate that the GVAE learns a smooth, meaningful latent space for arithmetic equations and molecules. Given this we perform optimization in this latent space using Bayesian optimization, inspired by the technique of Gómez-Bombarelli et al. (2016b). We demonstrate that the GVAE improves upon a previous character variational autoencoder, by selecting an arithmetic expression that matches the data nearly perfectly, and by finding novel molecules with better drug properties.

https://github.com/maxhodak/keras-molecules

4.1. Problems

We describe in detail the two sequence optimization problems we seek to solve. The first consists in optimizing the fit of an arithmetic expression. We are given a set of 100,000 randomly generated univariate arithmetic expressions from the following grammar:

```
S → S '+' T | S '*' T | S '/' T | T
T → '(' S ')' | 'sin(' S ')' | 'exp(' S ')'
T → 'x' | '1' | '2' | '3'
```

where S and T are non-terminals and the symbol | separates the possible production rules generated from each non-terminal. By parsing this grammar we can randomly generate strings of univariate arithmetic equations (functions of x) such as the following: $\texttt{sin(2)}$, $\texttt{x/(3+1)}$, $\texttt{2+x+sin(1/2)}$, and $\texttt{x/2*exp(x)/exp(2*x)}$. We limit the length of every selected string to have at most 15 production rules. Given this dataset we train both the CVAE and GVAE to learn a latent space of arithmetic expressions. We propose to perform optimization in this latent space of expressions to find an expression that best fits a fixed dataset. A common measure of best fit is the test MSE between the predictions made by a selected expression and the true data. In the generated expressions, the presence of exponential functions can result in very large MSE values. For this reason, we use as target variable $\log(1 + \text{MSE})$ instead of MSE.

For the second optimization problem, we follow (Gómez-Bombarelli et al., 2016b) and optimize the drug properties of molecules. Our goal is to maximize the water-octanol partition coefficient (logP), an important metric in drug design that characterizes the drug-likeness of a molecule. As in Gómez-Bombarelli et al. (2016b) we consider a penalized logP score that takes into account other molecular properties such as ring size and synthetic accessibility (Ertl & Schuffenhauer, 2009). The training data for the CVAE and GVAE models are 250,000 SMILES strings (Weininger, 1988) extracted at random from the ZINC database by Gómez-Bombarelli et al. (2016b). We describe the context-free grammar for SMILES strings that we use to train our GVAE in the supplementary material.

4.2. Visualizing the latent space

Arithmetic expressions. To qualitatively evaluate the smoothness of the VAE embeddings for arithmetic expressions, we attempt interpolating between two arithmetic expressions, as in Bowman et al. (2016). This is done by encoding two equations and then performing linear interpolation in the latent space. Results comparing the character and grammar VAEs are shown in Table 1. Although the character VAE smoothly interpolates between the text representation of equations, it passes through intermediate

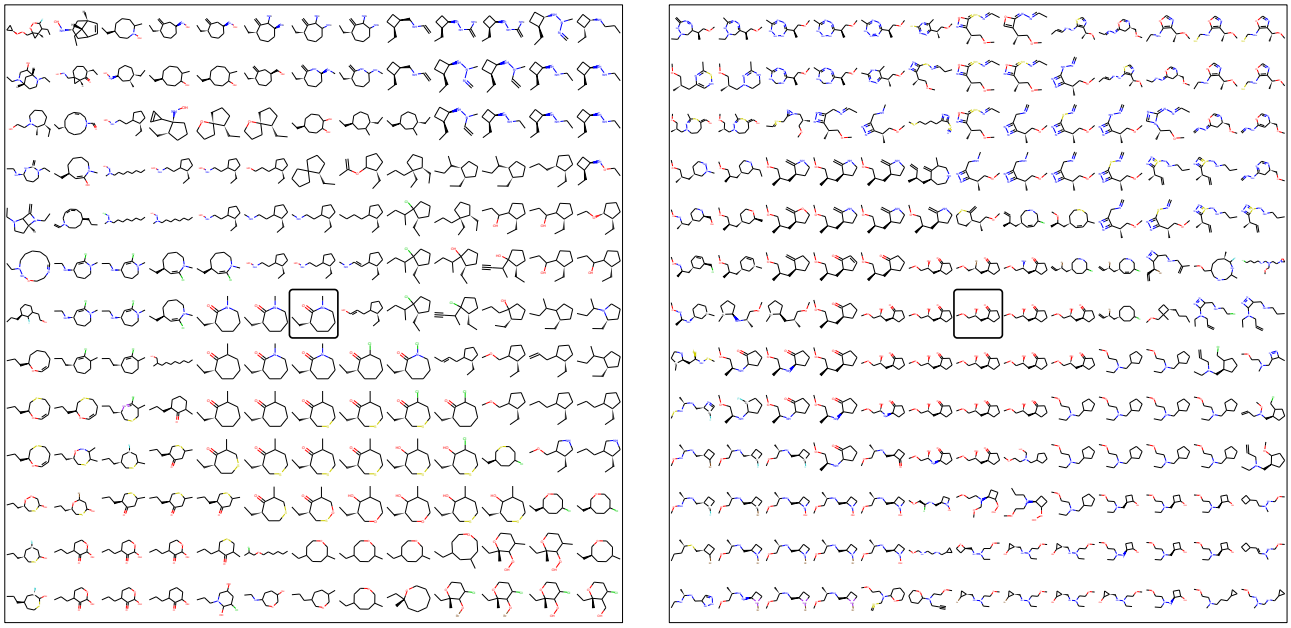

Figure 3. Searching the 56-dimensional latent space of the GVAE, starting at the molecule in the center.

points which do not decode to valid equations. In contrast, the grammar VAE also provides smooth interpolation *and* produces valid equations for any location in the latent space. A further exploration of a 2-dimensional latent space is shown in the appendix.

Molecules. We are interested if the GVAE produces a coherent latent space of molecules. To assess this we begin by encoding a molecule. We then generate 2 random orthogonal unit vectors in latent space (scaled down to only search the neighborhood of the molecules). Moving in combinations of these directions defines a grid and at each point in the grid we decode the latent vector 1000 times. We select the molecule that appears most often as the representative molecule. Figure 3 shows this latent space search surrounding two different molecules. Compare this to Figures 13-15 in Gómez-Bombarelli et al. (2016b). We note that in each plot of the GVAE the latent space is very smooth, in many cases moving from one grid point to another will only change a single atom in a molecule. In the CVAE (Gómez-Bombarelli et al., 2016b) we do not observe such fine-grained smoothness.

4.3. Bayesian optimization

We now perform a series of experiments using the autoencoders to produce novel sequences with improved properties. For this, we follow the approach proposed by Gómez-Bombarelli et al. (2016b) and after training the GVAE, we train an additional model to predict properties of sequences from their latent representation. To propose promising new sequences, we can start from the latent vector of an encoded sequence and then use the output of this predictor (including its gradient) to move in the latent space direction most likely to improve the property. The resulting new latent points can then be decoded into corresponding sequences.

In practice, measuring the property of each new sequence could be an expensive process. For example, the sequence could represent an organic photovoltaic molecule and the property could be the result of an expensive quantum mechanical simulation used to estimate the molecule's power-conversion efficiency (Hachmann et al., 2011). The sequence could also represent a program or expression which may be computationally expensive to evaluate. Therefore, ideally, we would like the optimization process to perform only a reduced number of property evaluations. For this, we use Bayesian optimization methods, which choose the next point to evaluate by maximizing an acquisition function that quantifies the benefit of evaluating the property at a particular location (Shahriari et al., 2016).

After training the GVAE, we obtain a latent feature vector for each sequence in the training data, given by the mean of the variational encoding distributions. We use these vectors and their corresponding property estimates to train a sparse Gaussian process (SGP) model with 500 inducing points (Snelson & Ghahramani, 2005), which is used to make predictions for the properties of new points in latent space. After training the SGP, we then perform 5 iterations of batch Bayesian optimization using the expected improvement (EI) heuristic (Jones et al., 1998). On each iteration, we select a batch of 50 latent vectors by sequen-

Character VAE	Grammar VAE
3*x+exp(3)+exp(1)	**3*x+exp(3)+exp(1)**
2*2+exp(3)+exp(1)	3*x+exp(3)+exp(1)
3*1+exp(3)+exp(2)	3*x+exp(x)+exp(1/2)
2*1+exp3)+exp(2)	2*x+exp(x)+exp(1/2)
2*3+(x)+exp(x*3)	2*x+(x)+exp(1*x)
2*x+(2)+exp(x*3)	2*x+(x)+exp(x*x)
2*x+(1)+exp(x*x)	**2*x+(1)+exp(x*x)**
3*x+exp(1)+(x+3)	**3*x+exp(1)+(x+3)**
3*x+exp(3)+(x*3)	3*x+exp(1)+(x+3)
3*1+exp(3)+(2*1)	2*3+exp(x)+(x)
3*x+exp(3)+(2*1)	2*3+x+(x+3)
2*1+exp(3)+(x*2)	2*3+x+(x/3)
2*x+exp3)+xx(3)	2*2+3+(x*3)
2*2+3+exp(x*3)	**2*2+3+exp(x*3)**
x+1+exp(1)+sin(1*2)	**x+1+exp(1)+sin(1*2)**
x+1+exp(1)+sin(1*2)	x+1+exp(1)+sin(1*2)
1+3+exp(x)+(i*1)	x/1+exp(x)+sin(x*2)
3+1+exp(2)+(1*1)	x/x+sin(x)+exp(x*2)
x+2+exp(x)+(2*3)	3*x+sin(x)+(x*3)
x*3+exp(3)+(3*2)	3*x+sin(3)+(3*3)
3*3+sin(3)+(3*3)	**3*3+sin(3)+(3*3)**
3*x+sin(2)+(x*x)	**3*x+sin(2)+(x*x)**
x*1+exp(x)+ex*3)	3*x+sin(2)+(x*x)
x*2+exp(x)+ex*x)	3*x+sin(2)+(3*x)
x*2+exp(x)+(x*1)	3*x+exp(2)+(3*3)
x*3+exp(x)+(x*3)	3*x+exp(2)+(3*3)
x*1+exp(x)+(2*2)	3*x+exp(2)+(2*2)
3*x+exp(2)+(2*2)	**3*x+exp(2)+(2*2)**

Table 1. Linear interpolation between two equations (in bold, at top and bottom of each cell). The character VAE often passes through intermediate strings which do not decode to a valid equation (shown in red). The grammar VAE makes subjectively smaller perturbations at each stage.

Table 2. Results finding best expression and molecule

Problem	Method	Frac. valid	Avg. score
Expressions	GVAE	**0.99±0.01**	**3.47 ±0.24**
	CVAE	0.86±0.06	4.75±0.25
Molecules	GVAE	**0.31±0.07**	**-9.57 ±1.77**
	CVAE	0.17±0.05	-54.66±2.66

tially maximizing the EI acquisition function. We use the Kriging Believer Algorithm to account for pending evaluations in the batch selection process (Cressie, 1990). That is, after selecting each new data point in the batch, we add that data point as a new inducing point in the sparse GP model with associated target variable equal to the mean of the GP predictive distribution at that point. Once a new batch of 50 latent vectors is selected, each point in the batch is transformed into its corresponding sequence using the decoder network in the GVAE. The properties of the newly generated sequences are then computed and the resulting data is added to the training set before retraining the SGP and starting the next BO iteration. Note that some of the new sequences will be invalid and consequently, it will not be possible to obtain their corresponding property estimate. In this case we fix the property to be equal to the worst value observed in the original training data.

Arithmetic expressions. Our goal is to see if we can find an arithmetic expression that best fits a fixed dataset.

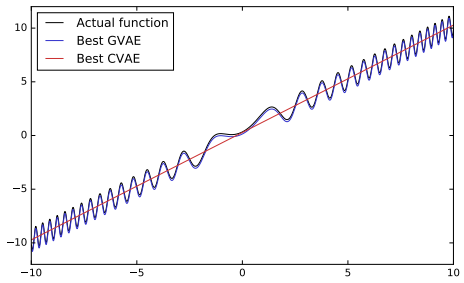

Figure 4. Plot of best expressions found by each method

Table 3. Best expressions found by each method

Method	#	Expression	Score
GVAE	1	$\mathtt{x}/1 + \sin(3) + \sin(\mathtt{x} * \mathtt{x})$	**0.04**
	2	$1/2 + (\mathtt{x}) + \sin(\mathtt{x} * \mathtt{x})$	**0.10**
	3	$\mathtt{x}/\mathtt{x} + (\mathtt{x}) + \sin(\mathtt{x} * \mathtt{x})$	**0.37**
CVAE	1	$\mathtt{x} * 1 + \sin(3) + \sin(3/1)$	0.39
	2	$\mathtt{x} * 1 + \sin(1) + \sin(2 * 3)$	0.40
	3	$\mathtt{x} + 1 + \sin(3) + \sin(3 + 1)$	0.40

Specifically, we generate this dataset by selecting 1000 input values, x, that are linearly-spaced between -10 and 10. We then pass these through our true function $1/3 + \mathtt{x} + \sin(\mathtt{x} * \mathtt{x})$ to generate the true target observations. We use Bayesian optimization (BO) as described above search for this equation. We run BO for 5 iterations and average across 10 repetitions of the process. Table 2 (rows 1 & 2) shows the results obtained. The third column in the table reports the fraction of arithmetic sequences found by BO that are valid. The GVAE nearly always finds valid sequences. The only cases in which it does not is when there are still non-terminals on the stack of the decoder upon reaching the maximum number of time-steps T_{max}, however this is rare. Additionally, the GVAE finds squences with better scores on average when compared with the CVAE.

Table 3 shows the top 3 expressions found by GVAE and CVAE during the BO search, together with their associated score values. Figure 4 shows how the best expression found by GVAE and CVAE compare to the true function. We note that the CVAE has failed to find the sinusoidal portion of the true expression, while the difference between the GVAE expression and the true function is negligible.

Molecules. We now consider the problem of finding new drug-like molecules. We perform 10 iterations of BO, and average results across 5 trials. Table 2 (rows 3 & 4) shows the overall BO results. In this problem, the GVAE produces about twice more valid sequences than the CVAE. The valid sequences produced by the GVAE also result in higher scores on average. The best found SMILES strings by each method and their scores are shown in Table 4; the

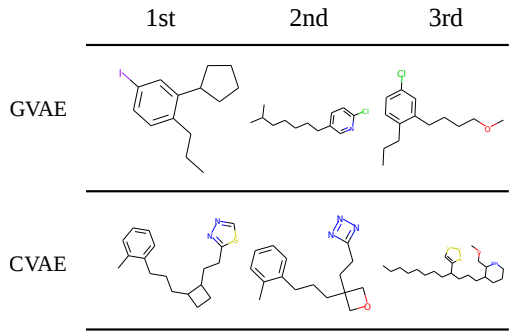

Figure 5. Plot of best molecules found by each method.

Table 4. Best molecules found by each method

Method	#	SMILE	Score
GVAE	1	CCCc1ccc(I)cc1C1CCC-c1	**2.94**
	2	CC(C)CCCCCc1ccc(Cl)nc1	**2.89**
	3	CCCc1ccc(Cl)cc1CCCCOC	**2.80**
CVAE	1	Cc1ccccc1CCCC1CCC1CCc1nncs1	1.98
	2	Cc1ccccc1CCCC1(COC1)CCc1nnn1	1.42
	3	CCCCCCCCC(CCCC212CCCnC1COC)c122csss1	1.19

molecules themselves are plotted in Figure 5.

4.4. Predictive performance of latent representation

We now perform a series of experiments to evaluate the predictive performance of the latent representations found by each autoencoder. For this, we use the sparse GP model used in the previous Bayesian optimization experiments and look at its predictive performance on a left-out test set with 10% of the data, where the data is formed by the latent representation of the available sequences (these are the inputs to the sparse GP model) and the associated properties of those sequences (these are the outputs in the sparse GP model). Table 5 show the average test RMSE and test log-likelihood for the GVAE and the CVAE across 10 different splits of the data for the expressions and for the molecules. This table shows that the GVAE produces latent features that yield much better predictive performance than those produced by the CVAE.

5. Related Work

Parse trees have been used to learn continuous representations of text in recursive neural network models (Socher et al., 2013; Irsoy & Cardie, 2014; Paulus et al., 2014). These models learn a vector at every non-terminal in the parse tree by recursively combining the vectors of child nodes. Recursive autoencoders learn these representations by minimizing the reconstruction error between true child vectors and those predicted by the parent (Socher et al., 2011b;a). Recently, Allamanis et al. (2016) learn representations for symbolic expressions from their parse trees.

Table 5. Test Log-likelihood (LL) and RMSE for the sparse GP predictions of penalized LogP score from the latent space

Objective	Method	Expressions	Molecules
LL	GVAE	**-1.320±0.001**	**-1.739 ±0.004**
	CVAE	-1.397±0.003	-1.812±0.004
RMSE	GVAE	**0.884 ±0.002**	**1.404 ±0.006**
	CVAE	0.975±0.004	1.504±0.006

Importantly, all of these methods are discriminative and do not learn a generative latent space.

Learning arithmetic expressions to fit data, often called *symbolic regression*, are generally based on genetic programming (Willis et al., 1997) or other computationally demanding evolutionary algorithms to propose candidate expressions (Schmidt & Lipson, 2009). Alternatives include running particle MCMC inference to estimate a Bayesian posterior over parse trees (Perov & Wood, 2016).

In molecular design, searching for new molecules is traditionally done by sifting through large databases of potential molecules and then subjecting them to a virtual screening process (Pyzer-Knapp et al., 2015; Gómez-Bombarelli et al., 2016a). These databases are too large to search via exhaustive enumeration, and require novel stochastic search algorithms tailored to the domain (Virshup et al., 2013; Rupakheti et al., 2015). Segler et al. (2017) fit a recurrent neural network to chemicals represented by SMILES strings, however their goal is more akin to density estimation; they learn a simulator which can sample proposals for novel molecules, but it is not otherwise used as part of an optimization or inference process itself. Our work most closely resembles Gómez-Bombarelli et al. (2016b) for novel molecule synthesis, in that we also learn a latent variable model which admits a continuous representation of the domain. However, both Segler et al. (2017) and Gómez-Bombarelli et al. (2016b) use character-level models for molecules.

6. Discussion

Empirically, it is clear that representing molecules and equations by way of their parse tree outperforms text-based representations. We believe this approach will be broadly useful for representation learning, inference, and optimization in any domain which can be represented as text in a context-free language.

Acknowledgements

This work was supported by The Alan Turing Institute under the EPSRC grant EP/N510129/1.

References

Allamanis, Miltiadis, Chanthirasegaran, Pankajan, Kohli, Pushmeet, and Sutton, Charles. Learning continuous semantic representations of symbolic expressions. *arXiv preprint arXiv:1611.01423*, 2016.

Baker, James K. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65(S1):S132–S132, 1979.

Booth, Taylor L and Thompson, Richard A. Applying probability measures to abstract languages. *IEEE transactions on Computers*, 100(5):442–450, 1973.

Bowman, Samuel R, Vilnis, Luke, Vinyals, Oriol, Dai, Andrew M, Jozefowicz, Rafal, and Bengio, Samy. Generating sentences from a continuous space. *CoNLL 2016*, pp. 10, 2016.

Cho, Kyunghyun, Van Merriënboer, Bart, Gulcehre, Caglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Cressie, Noel. The origins of kriging. *Math. Geol.*, 22(3): 239–252, 1990.

Ertl, Peter and Schuffenhauer, Ansgar. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of cheminformatics*, 1(1):8, 2009.

Gatys, Leon A, Ecker, Alexander S, and Bethge, Matthias. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

Gaunt, Alexander L, Brockschmidt, Marc, Singh, Rishabh, Kushman, Nate, Kohli, Pushmeet, Taylor, Jonathan, and Tarlow, Daniel. Terpret: A probabilistic programming language for program induction. *arXiv preprint arXiv:1608.04428*, 2016.

Gómez-Bombarelli, Rafael, Aguilera-Iparraguirre, Jorge, Hirzel, Timothy D, Duvenaud, David, Maclaurin, Dougal, Blood-Forsythe, Martin A, Chae, Hyun Sik, et al. Design of efficient molecular organic light-emitting diodes by a high-throughput virtual screening and experimental approach. *Nature Materials*, 15(10):1120–1127, 2016a.

Gómez-Bombarelli, Rafael, Duvenaud, David, Hernández-Lobato, José Miguel, Aguilera-Iparraguirre, Jorge, Hirzel, Timothy D, Adams, Ryan P, and Aspuru-Guzik, Alán. Automatic chemical design using a data-driven continuous representation of molecules. *arXiv preprint arXiv:1610.02415*, 2016b.

Graves, Alex. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Hachmann, J., Olivares-Amaya, R., Atahan-Evrenk, S., Amador-Bedolla, C., Sanchez-Carrera, R. S., Gold-Parker, A., Vogt, L., Brockway, A. M., and Aspuru-Guzik, A. The Harvard Clean Energy Project: Large-Scale Computational Screening and Design of Organic Photovoltaics on the World Community Grid. *J. Phys. Chem. Lett.*, 2(17):2241–2251, sep 2011.

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

Hopcroft, John E, Motwani, Rajeev, and Ullman, Jeffrey D. *Introduction to Automata theory, languages, and computation*. 2006.

Irsoy, Ozan and Cardie, Claire. Deep recursive neural networks for compositionality in language. In *NIPS*, pp. 2096–2104, 2014.

James, Craig A, Vandermeersch, T, and Dalke, A. Opensmiles specification, 2015.

Jaques, Natasha, Gu, Shixiang, Turner, Richard E, and Eck, Douglas. Tuning recurrent neural networks with reinforcement learning. *arXiv preprint arXiv:1611.02796*, 2016.

Johnson, Mark, Griffiths, Thomas L, Goldwater, Sharon, et al. Adaptor grammars: A framework for specifying compositional nonparametric bayesian models. *Advances in neural information processing systems*, 19: 641, 2007.

Jones, Donald R, Schonlau, Matthias, and Welch, William J. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13 (4):455–492, 1998.

Kalchbrenner, Nal, Grefenstette, Edward, and Blunsom, Phil. A convolutional neural network for modelling sentences. 2014.

Kernighan, Brian W, Ritchie, Dennis M, and Ejeklint, Per. *The C programming language*, volume 2. Prentice-Hall Englewood Cliffs, 1988.

Kingma, Diederik P and Welling, Max. Auto-encoding variational Bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

Kusner, Matt J and Hernández-Lobato, José Miguel. Gans for sequences of discrete elements with the gumbel-softmax distribution. *arXiv:1611.04051*, 2016.

Paulus, Romain, Socher, Richard, and Manning, Christopher D. Global belief recursive neural networks. In *Advances in Neural Information Processing Systems*, pp. 2888–2896, 2014.

Perov, Yura and Wood, Frank. Automatic sampler discovery via probabilistic programming and approximate bayesian computation. In *International Conference on Artificial General Intelligence*, pp. 262–273, 2016.

Pyzer-Knapp, Edward O, Suh, Changwon, Gómez-Bombarelli, Rafael, Aguilera-Iparraguirre, Jorge, and Aspuru-Guzik, Alán. What is high-throughput virtual screening? a perspective from organic materials discovery. *Annual Review of Materials Research*, 45:195–216, 2015.

Radford, Alec, Metz, Luke, and Chintala, Soumith. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Rezende, Danilo Jimenez, Mohamed, Shakir, and Wierstra, Daan. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Riedel, Sebastian, Bosnjak, Matko, and Rocktäschel, Tim. Programming with a differentiable forth interpreter. *CoRR, abs/1605.06640*, 2016.

Rupakheti, Chetan, Virshup, Aaron, Yang, Weitao, and Beratan, David N. Strategy to discover diverse optimal molecules in the small molecule universe. *Journal of chemical information and modeling*, 55(3):529–537, 2015.

Schmidt, Michael and Lipson, Hod. Distilling free-form natural laws from experimental data. *Science*, 324 (5923):81–85, 2009.

Segler, Marwin HS, Kogej, Thierry, Tyrchan, Christian, and Waller, Mark P. Generating focussed molecule libraries for drug discovery with recurrent neural networks. *arXiv preprint arXiv:1701.01329*, 2017.

Shahriari, Bobak, Swersky, Kevin, Wang, Ziyu, Adams, Ryan P, and de Freitas, Nando. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

Snelson, Edward and Ghahramani, Zoubin. Sparse Gaussian processes using pseudo-inputs. In *NIPS*, pp. 1257–1264, 2005.

Socher, Richard, Huang, Eric H, Pennington, Jeffrey, Ng, Andrew Y, and Manning, Christopher D. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *NIPS*, volume 24, pp. 801–809, 2011a.

Socher, Richard, Pennington, Jeffrey, Huang, Eric H, Ng, Andrew Y, and Manning, Christopher D. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the conference on empirical methods in natural language processing*, pp. 151–161. Association for Computational Linguistics, 2011b.

Socher, Richard, Perelygin, Alex, Wu, Jean Y, Chuang, Jason, Manning, Christopher D, Ng, Andrew Y, Potts, Christopher, et al. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, pp. 1642. Citeseer, 2013.

Upchurch, Paul, Gardner, Jacob, Bala, Kavita, Pless, Robert, Snavely, Noah, and Weinberger, Kilian. Deep feature interpolation for image content changes. *arXiv preprint arXiv:1611.05507*, 2016.

Virshup, Aaron M, Contreras-García, Julia, Wipf, Peter, Yang, Weitao, and Beratan, David N. Stochastic voyages into uncharted chemical space produce a representative library of all possible drug-like compounds. *Journal of the American Chemical Society*, 135(19):7296–7303, 2013.

Weininger, David. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci.*, 28(1):31–36, 1988.

Willis, M-J, Hiden, Hugo G, Marenbach, Peter, McKay, Ben, and Montague, Gary A. Genetic programming: An introduction and survey of applications. In *Genetic Algorithms in Engineering Systems*, pp. 314–319. IET, 1997.

Appendix

A. Grammars for equations and SMILES

The grammar for the single-variable equations includes 3 binary operators, 2 unary operators, 3 constants, and grouping symbols; the start symbol is `S`. Training data for the VAE came by generating 100,000 different equations with parse tree depth less than 7, corresponding to equations which can be produced using up to 15 production rule applications.

```
S → S '+' T | S '*' T | S '/' T | T
T → '(' S ')' | 'sin(' S ')' | 'exp(' S ')' | 'x' | '1' | '2' | '3'
```

The grammar for SMILES is based on the official OPENSMILES specification (Weininger, 1988), and starts with `smiles`.

```
smiles → chain
atom → bracket_atom | aliphatic_organic | aromatic_organic
aliphatic_organic → 'B' | 'C' | 'N' | 'O' | 'S' | 'P' | 'F' | 'I' | 'Cl' | 'Br'
aromatic_organic → 'c' | 'n' | 'o' | 's'
bracket_atom → '[' BAI ']'
BAI → isotope symbol BAC | symbol BAC | isotope symbol | symbol
BAC → chiral BAH | BAH | chiral
BAH → hcount BACH | BACH | hcount
BACH → charge class | charge | class
symbol → aliphatic_organic | aromatic_organic
isotope → DIGIT | DIGIT DIGIT | DIGIT DIGIT DIGIT
DIGIT → '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8'
chiral → '@' | '@@'
hcount → 'H' | 'H' DIGIT
charge → '-' | '-' DIGIT | '-' DIGIT DIGIT | '+' | '+' DIGIT | '+' DIGIT DIGIT
bond → '-' | '=' | '#' | '/' | '\'
ringbond → DIGIT | bond DIGIT
branched_atom → atom | atom RB | atom BB | atom RB BB
RB → RB ringbond | ringbond
BB → BB branch | branch
branch → '(' chain ')' | '(' bond chain ')'
chain → branched_atom | chain branched_atom | chain bond branched_atom
```

B. Network structure

We briefly overview recent sequence modeling advances which inform our encoder and decoder models. An encoder $q_\phi(\mathbf{z}|\mathbf{X})$ takes a sequence of T timesteps $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$ as input and returns a distribution over real-valued vectors z, whereas the decoder $p_\theta(\mathbf{x}|\mathbf{z})$ takes a real-valued vector z as input and generates a distribution over sequences themselves $\mathbf{X}$. We use the same encoder and decoder as Gómez-Bombarelli et al. (2016b), inspired by Bowman et al. (2016), with a few modifications as described in Section 3. In general, we encode the data as sequences of one-hot vectors and apply a series of one-dimensional convolutions to the sequence data (Kalchbrenner et al., 2014). These are followed by fully-connected layers that predict the mean and variance parameters of a Gaussian distribution $q_\phi(\mathbf{z}|\mathbf{x})$. To decode, we use recurrent neural network models for sequences (Graves, 2013; Cho et al., 2014), to output discrete probabilities over symbols at each timestep to define $p_\theta(\mathbf{x}|\mathbf{z})$. For more architecture details see Gómez-Bombarelli et al. (2016b).

Table 6. Reconstruction accuracy and sample validity results.

Method	% Reconstruct	% Prior Valid
GVAE	**53.7**	**7.2**
CVAE	44.6	0.70

C. Additional experiments

Molecule reconstruction & validity. We characterize how well the VAE models over molecules are able to reconstruct input sequences from their corresponding latent representations and to also decode valid sequences when sampling from the prior in latent space. Comparisons of full reconstruction accuracy for both the character and grammar VAEs are shown in Table 6. To compute reconstruction error we start with 5000 true molecules from a hold-out set. For each molecule we encode it 10 times, and we decode each encoding 100 times (as encoding and decoding are stochastic). This results in 1000 decoded molecules for each of the 5000 input molecules. We compute the average of these 1000 decodings that are

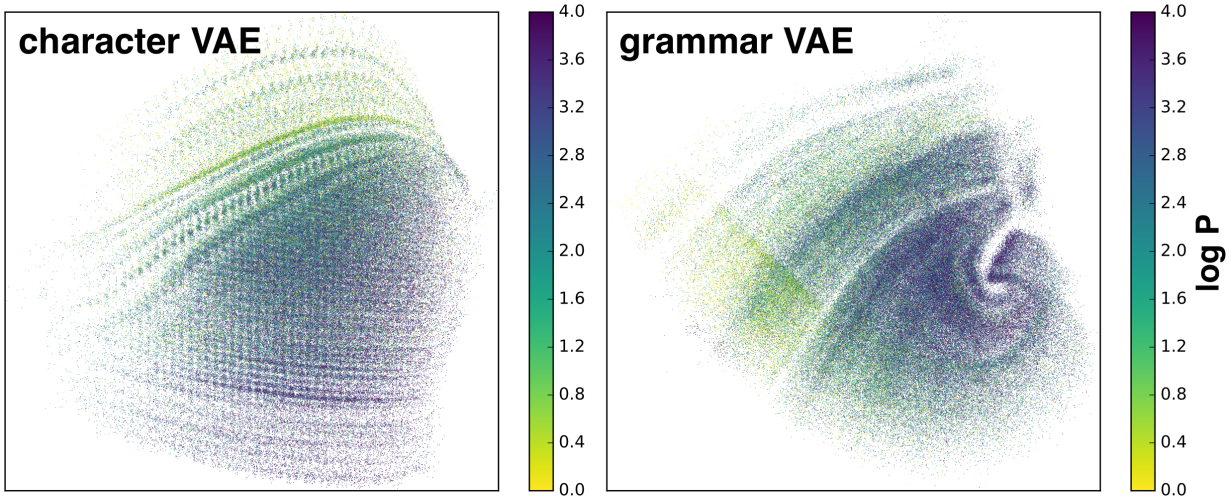

Figure 6. The logP values of a 2-dimensional character and grammar VAE. The grammar VAE leads to a low-dimensional latent space which is visually smother with respect to the property of interest.

identical to the input molecule. We then average these averages across all 5000 inputs to get the percentage of molecules that reconstruct out of the $5,000,000$ attempts. To compute the percentage prior validity we sample 1000 latent points from the prior distribution $p(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$. We decode each of these points 500 times and test which of the decoded SMILES strings correspond to valid molecules. We average across all 1000 points and 500 trials to yield the percentages in Table 6. These results clearly indicate that the proposed GVAE has higher reconstruction accuracy, and produces a higher proportion of valid sequences when sampling from the prior.

LogP Visualization. To visualize the latent space of the VAEs on molecules with train a CVAE and GVAE on the ZINC dataset (Gómez-Bombarelli et al., 2016b) with a 2-dimensional latent space. We plot the training set colored by the logP values of the molecules in Figure 6. We note that the CVAE seems to have higher logP values (corresponding to molecules with better drug properties) in the lower portion of the latent space. The GVAE on the other hand concentrates molecules with high logP in a small region of latent space. We suspect this makes Bayesian optimization for molecules with high logP much easier in the GVAE.