# 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 $\mathbf{z}$ 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 to incorporate a grammar as described in Section 3. We begin by encoding the data as sequences of one-hot vectors. We then pass these vectors through our encoder network which consists of 3 layers of one-dimensional convolutions (Kalchbrenner et al., 2014). We then flatten the resulting sequence into a vector and pass it through a fully connected layer. We then pass the resulting vector through two separate fully connected layers to produce the mean and variance of the latent distribution $q_\phi(\mathbf{z}|\mathbf{X})$ over $\mathbf{z}$. To decode, we sample a $\mathbf{z}$ from this distribution and pass it through a fully connected layer. We then repeat the output of this layer $T_{max}$ times, producing a matrix of dimension $H \times T_{max}$, where $H$ is the dimensionality of $\mathbf{z}$. This can be thought of as a sequence over $T_{max}$ timesteps. We then pass this matrix through 3 layers of GRU recurrent units (Cho et al., 2014), and finally through a fully connected layer, whose weights are repeated at each timestep. The result are the logit vectors $\mathbf{F}$.

## C. Implementation details

We make two implementation changes to the standard VAE. We begin by reconsidering the ELBO as the sum of a reconstruction loss term and a negative KL-divergence term (Kingma & Welling, 2014),

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

The first term ensures that the latent representation $\mathbf{z}$ serves as an accurate 'encoding' or 'compression' of the original data $\mathbf{X}$. The second term serves to regularize the latent representation so that the latent space is not too complex. Combined together, these terms are aimed at producing a latent space which is the most efficient encoding of the data $\mathbf{X}$. However, when optimizing this ELBO in the usual way, we noticed that the VAE had the tendency to ignore the first term and simply

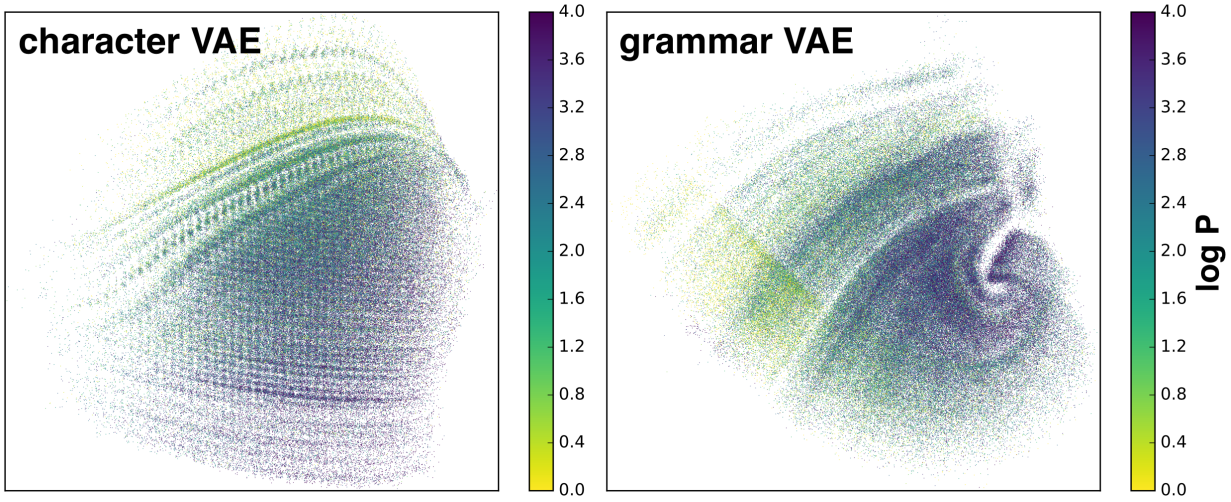


*Figure 6.* The logP values of a 2-dimensional character and grammar VAE. The grammar VAE leads to a low-dimensional latent space which is visually smother with respect to the property of interest.

try to optimize the second term by trying to have the encoder $q_\phi(\mathbf{z}|\mathbf{X})$ match the prior distribution $p(\mathbf{z})$. Thus, the learned latent representation $\mathbf{z}$ contains essentially no information about $\mathbf{X}$. We believe this happened because our decoder is such a powerful recurrent neural network, that it easily produces decodings that 'appear' similar enough to the data $\mathbf{X}$. We note that this behavior has been observed in other recent work (Zhao et al., 2017). To fix this we made two small modifications, inspired by the character VAE on which our code is based[4]. First, we scale the standard deviation of the encoding distribution $q_\phi(\mathbf{z}|\mathbf{X})$ by $0.01$. We found that this helped the VAE produce more consistent encodings $\mathbf{z}$ for data $\mathbf{X}$, thus making the latent-space better resemble the data. Second, we divide the KL-divergence term by the number of latent dimensions $H$. This effectively downweights the importance of the KL term so the VAE focusses more of an effort at improving reconstruction.

*Table 6.* Reconstruction accuracy and sample validity results.

| Method | % Reconstruct | % Prior Valid |
|---|---|---|
| GVAE | **53.7** | **25.9** |
| CVAE | 44.6 | 0.001 |

## D. 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 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.

[4]https://github.com/maxhodak/keras-molecules

**LogP Visualization.** To visualize the latent space of the VAEs on molecules we 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 disperse molecules with higher logP values (corresponding to molecules with better drug properties) throughout a large region in the lower half 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.