LEARNING A GENERATIVE MODEL FOR VALIDITY IN COMPLEX DISCRETE STRUCTURES

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep generative models have been successfully used to learn representations for high-dimensional discrete spaces by representing discrete objects as sequences, for which powerful sequence-based deep models can be employed. Unfortunately, these techniques are significantly hindered by the fact that these generative models often produce invalid sequences: sequences which do not represent any underlying discrete structure. As a step towards solving this problem, we propose to learn a deep recurrent *validator* model, which can estimate whether a partial sequence can function as the beginning of a full, valid sequence. This model not only discriminates between valid and invalid sequences, but also provides insight as to how individual sequence elements influence the validity of discrete objects. To learn this model we propose a reinforcement learning approach, where an oracle which can evaluate validity of complete sequences provides a sparse reward signal. We believe this is a key step toward learning generative models that faithfully produce valid discrete objects, and demonstrate its effectiveness in evaluating the validity of Python 3 source code for mathematical expressions, and generating SMILES string representations of molecules.

1 INTRODUCTION

Deep generative modeling has seen many successful recent developments, such as producing realistic images from noise (Radford et al., 2015) and creating artwork (Gatys et al., 2016). We find particularly promising the opportunity to leverage deep generative models for search in high-dimensional discrete spaces (Gómez-Bombarelli et al., 2016b; Kusner et al., 2017). Discrete search is at the heart of problems in drug discovery (Gómez-Bombarelli et al., 2016a), natural language processing (Bowman et al., 2016; Guimaraes et al., 2017), and symbolic regression (Kusner et al., 2017).

This approach to attacking these discrete search problems works by 'lifting' the search from discrete space to continuous space, via an autoencoder (Rumelhart et al., 1985). An autoencoder learns two mappings: 1) a mapping from discrete space to continuous space called an *encoder*; and 2) a reverse mapping from continuous space back to discrete space called a *decoder*. The discrete space is presented to the autoencoder in as a sequence in some formal language — for example, in Gómez-Bombarelli et al. (2016b) molecules are encoded as SMILES strings — powerful sequential models (e.g., LSTMs (Hochreiter & Schmidhuber, 1997) GRUs (Cho et al., 2014), DCNNs (Kalchbrenner et al., 2014)) can then be applied to the string representation. Unfortunately, most sequences using the same characters as those in a valid SMILES string are not themselves actually valid SMILES: the string does not actually represent any real molecule structure. When employing these models as encoders and decoders, generation of invalid sequences is still possible, and currently this happens frequently. Kusner et al. (2017) aimed to fix this by using a context-free grammar to rule out generating syntactically invalid sequences. However, the grammar only describes syntactic constraints and cannot enforce semantic constraints, so certain invalid sequences can still be generated.

In this paper, we propose a generative approach to modeling validity that can learn the validity constraints of a given discrete space. In section 2, we show how concepts from reinforcement learning may be used to define a suitable generative model and how this generative model can be approximated using Bayesian recurrent neural networks (Gal & Ghahramani, 2016). As the length of sequences grows, as well as the number of possible elements, it quickly becomes impossible to observe more than a vanishingly small subset of all possible sequences during training. To circumvent this, in

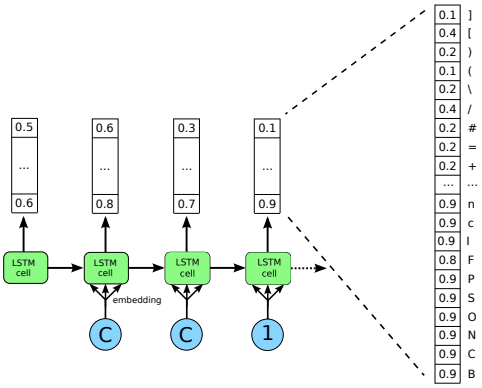

Figure 1: The LSTM model used to approximate the Q-function. A hypothetical logistic function activation is shown for each character in $\mathcal{C}$. Here the set of characters is the SMILES alphabet and we use the first 3 characters of the molecule in figure 3 as the input example. The initial character is predicted from the first hidden state, and the LSTM continues until the end of the sequence.

section 3 we describe methods for proposing high-quality candidate training sequences, including a novel approach to active learning for sequential tasks inspired by classic mutual-information-based approaches (Houlsby et al., 2011; Hernández-Lobato et al., 2014), and a data augmentation process based on applying minimal perturbations to known-valid sequences. This allows us to learn the proposed complex generative model with only a minimal number of examples. We test the quality of the the learned validity models by a series of experiments where we learn to discriminate syntactically and semantically valid sequences in two domains: mathematical expressions in the Python programming language, and the SMILES string representation of molecules.

2 A MODEL FOR SEQUENCE VALIDITY

To formalise the problem we denote the set of discrete sequences of length T by $\mathcal{X} = \{(x_1, \ldots, x_T) : x_t \in \mathcal{C}\}$, where $\mathcal{C} = \{1, \ldots, C\}$ is an alphabet of size C, individual sequences in $\mathcal{X}$ are denoted $x_{1:T}$. We assume the availability of a *validator* $v \colon \mathcal{X} \to \{0, 1\}$, an oracle which can tell us whether a given sequence is valid. Here, it is important to note that such a validator gives very sparse feedback: it can only be evaluated on a *complete* sequence. Examples of such validators are compilers for programming languages (which can identify syntax and type errors), or chemoinformatics software for parsing SMILES strings (which identify violations of valence constraints). Running the standard validity checker $v(x_{1:T})$ on a partial sequence or subsequence (e.g., the first $t < T$ characters of a computer program) does not in general provide any indication as to whether the final complete program of length T is also valid.

We aim to obtain a generative model for the sequence set $\mathcal{V} = \{x_{1:T} \in \mathcal{X} : v(x_{1:T}) = 1\}$, that is, the subset of valid sequences in $\mathcal{X}$. To achieve this, rather than only having a access to the validator $v(x_{1:T})$ which operates on full sequences, we would ideally like to be able to query a more-informative function $\tilde{v}(x_{1:t})$ which operates on prefixes $x_{1:t}$ of a hypothetical longer sequence $x_{1:T}$. We define $\tilde{v}(x_{1:t})$ as

$$\tilde{v}(x_{1:t}) = \begin{cases} 1 & \text{if there exists a suffix } x_{t+1:T} \text{ such that } v(\mathrm{c}[x_{1:t}, x_{t+1:T}]) = 1, \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $\mathrm{c}[x_{1:t}, x_{t+1:T}]$ concatenates a prefix and a suffix to form a complete sequence. The function $\tilde{v}(x_{1:t})$ can be used to determine whether a given prefix can ever successfully yield a valid outcome. Note that we are indifferent to *how many* suffixes yield valid sequences. If we had access to $\tilde{v}(x_{1:t})$, we could create a generative model for $\mathcal{V}$ which would form sequences from left to right, a single character at a time, using $\tilde{v}(x_{1:t})$ to provide early feedback as to which of the next character choices will surely not lead to a "dead end" from which no valid sequence can be produced.

We frame the problem of obtaining a generative model for $\mathcal{V}$ as a Markov decision process (Sutton & Barto, 1998), and train a reinforcement learning agent to select characters sequentially in a manner which avoids producing invalid sequences. At time $t = 1, \ldots, T$, an agent is in state $x_{<t} = x_{1:t-1}$ and can take actions $x_t \in \mathcal{C}$. At the end of an episode, following action x_T, the agent receives a reward of $v(x_{1:T})$. Since in practice we are only able to evaluate $v(x_{1:T})$ in a meaningful way on complete sequences, the agent does not receive any reward at any of the intermediate steps $t < T$. The optimal Q-function $Q^\star(s, a)$ (Watkins, 1989), a function of a state s and an action a, represents

the expected reward of an agent following an optimal policy which takes action a at state s. This optimal Q-function assigns value 1 to actions $a = x_t$ in state $s = x_{<t}$ for which there exists a suffix $x_{t+1:T}$ such that $\mathrm{c}[x_{1:t}, x_{t+1:T}] \in \mathcal{V}$, and value 0 to all other state/action pairs. This behaviour exactly matches the desired prefix validator in (1), that is, $Q^\star(x_{<t}, x_t) = \tilde{v}(x_{1:t})$.

Having access to $Q^\star$ would allow us to obtain a generative model for $\mathcal{V}$. In particular, an agent following any optimal policy $\pi^\star(x_{<t}) = \mathrm{argmax}_{x_t \in \mathcal{C}} Q^\star(x_{<t}, x_t)$, will always generate valid sequences. If we sample uniformly at random across all optimal actions at each time $t = 1, \ldots, T$, we obtain the joint distribution given by

$$p(x_{1:T}) = \prod_{t=1}^{T} \frac{Q^\star(x_{<t}, x_t)}{Z(x_{<t})}, \tag{2}$$

where $Z(x_{<t}) = \sum_{x_t} Q^\star(x_{<t}, x_t)$ are the per-timestep normalisation constants. This distribution allows us to sample sequences $x_{1:T}$ in a straightforward manner by sequentially selecting characters $x_t \in \mathcal{C}$ given the previously selected ones in $x_{<t}$.

We focus on learning an approximation to (2). For this, we use recurrent neural networks, which have recently shown remarkable empirical success in the modelling of sequencial data, e.g., in natural language processing applications (Sutskever et al., 2014). We approximate the optimal Q-function with a long-short term memory (LSTM) model Hochreiter & Schmidhuber (1997) that includes one output unit per character in $\mathcal{C}$, with each output unit using a logistic activation function (see figure 1). We denote by $\mathrm{y}(x_t|x_{<t}, w)$ the value at time t of the LSTM output unit corresponding to character x_t when the network weights are w the input is the sequence $x_{<t}$. We interpret the neural network output $\mathrm{y}(x_t|x_{<t}, w)$ as $p(Q^\star(x_{<t}, x_t) = 1)$, that is, as the probability that action x_t can yield a valid sequence given that the current state is $x_{<t}$. We consider that the prediction errors of the LSTM model are independent. Therefore, a sequence $x_{1:T}$ will be valid according to our model if every action during the sequence generation process is permissible, that is, if $Q^\star(x_{<t}, x_t) = 1$ for $t = 1, \ldots, T$. Similarly, we consider that the sequence $x_{1:T}$ will be invalid if at least one action during the sequence generation process is not valid[1] that is, if $Q^\star(x_{<t}, x_t) = 0$ at least once for $t = 1, \ldots, T$. This specifies the following log-likelihood function given a training set $\mathcal{D} = \{(x^n_{1:T}, y_n)\}_{n=1}^N$ of sequences $x^n_{1:T} \in \mathcal{X}$ and corresponding labels $y_n = v(x_{1:T})$:

$$\mathcal{L}(w|\mathcal{D}) = \sum_{n=1}^{N} \left\{ y_n \log p(y_n = 1|x^n_{1:T}, w) + (1 - y_n) \log p(y_n = 0|x^n_{1:T}, w) \right\} , \tag{3}$$

where, as described above, we define

$$p(y_n = 1|x^n_{1:T}, w) = \prod_{t=1}^{T} \mathrm{y}(x_t|x_{<t}, w) , \qquad p(y_n = 0|x^n_{1:T}, w) = 1 - \prod_{t=1}^{T} \mathrm{y}(x_t|x_{<t}, w) , \tag{4}$$

according to our model's predictions. The log-likelihood (3) can be optimised using backpropagation and stochastic gradient descent. Let $\hat{w}$ be the resulting maximizer. Then, when $\mathcal{D}$ is very large and the LSTM model has sufficient capacity, it can be shown that $\mathrm{y}(x_t|x_{<t}, \hat{w}) \approx Q^\star(x_{<t}, x_t)$ for all x_t and for all states $x_{<t}$ that can be reached by following any possible optimal policy $\pi^\star(x_{<t})$. This result is independent of the sampling distribution used to construct $\mathcal{D}$.

Instead of directly maximising (3), we can follow a Bayesian approach to obtain estimates of uncertainty in the predictions of our LSTM model. For this, we can introduce dropout layers which stochastically zero-out units in the input and hidden layers of the LSTM model according to a Bernoulli distribution (Gal & Ghahramani, 2016). Under the assumption of a Gaussian prior $p(w)$ over weights, the resulting stochastic process yields an implicit approximation $q(w)$ to the posterior distribution $p(w|\mathcal{D}) \propto \exp(\mathcal{L}(w|\mathcal{D}))p(w)$.

3 ONLINE GENERATION OF SYNTHETIC TRAINING DATA

One critical aspect of learning w as described above is how to generate the training set $\mathcal{D}$ in a sensible manner. A naïve approach could be to draw elements from $\mathcal{X}$ uniformly at random. However, in

[1] Note that, once $Q^\star(x_{<t}, x_t)$ is zero, all the following values of $Q^\star(x_{<t}, x_t)$ in that sequence will be irrelevant to us. Therefore, we can safely assume that a sequence is invalid if $Q^\star(x_{<t}, x_t)$ is zero at least once in that sequence.

many cases, $\mathcal{X}$ contains only a tiny fraction of valid sequences and the uniform sampling approach produces extremely unbalanced sets which are useless in practice. While rejection sampling can be used to increase the number of positive samples, the resulting additional cost makes such an alternative infeasible in most practical cases. The problem gets typically worse as the sequence length T increases since $|\mathcal{X}|$ always grow as $|\mathcal{C}|^T$, while $|\mathcal{V}|$ will typically grow at a lower rate.

We employ two approaches for artificially constructing balanced sets that permit learning these models in far fewer samples than $|\mathcal{C}|^T$. In settings where we do not have a corpus of known valid sequences, Bayesian active learning can automatically construct the training set $\mathcal{D}$. These methods work by iteratively selecting sequences in $\mathcal{X}$ that are maximally informative about the model parameters w given the data collected so far (MacKay, 1992). When we do have a set of known valid sequences, we can use these to seed a process for generating balanced sets by applying random perturbations to valid sequences.

3.1 Active learning

Let $x_{1:T}$ denote an arbitrary sequence and let y be the unknown binary label indicating whether $x_{1:T}$ is valid or not. Our model's predictive distribution for y, that is, $p(y|x_{1:T}, w)$ is given by (4). The amount of information on w that we expect to gain by labeling and adding $x_{1:T}$ to $\mathcal{D}$ can be measured in terms of the expected reduction in the entropy of the posterior distribution $p(w|\mathcal{D})$. That is,

$$\alpha(x_{1:T}) = \mathrm{H}[p(w|\mathcal{D})] - \mathbb{E}_{p(y|x_{1:T},w)}\mathrm{H}[p(w|\mathcal{D} \cup (x_{1:T}, y)]\,, \tag{5}$$

where $\mathrm{H}(\cdot)$ computes the entropy of a distribution. This formulation of the entropy-based active learning criterion is, however, difficult to approximate because it requires to work with the entropy of $p(w|\mathcal{D})$ which is intractable. To obtain a simpler expression we follow Houlsby et al. (2011) and note that $\alpha(x_{1:T})$ is equal to the mutual information between y and w given $x_{1:T}$ and $\mathcal{D}$:

$$\alpha(x_{1:T}) = \mathrm{H}\{\mathbb{E}_{p(w|\mathcal{D})}[p(y|x_{1:T}, w)]\} - \mathbb{E}_{p(w|\mathcal{D})}\{\mathrm{H}[p(y|x_{1:T}, w)]\}\,, \tag{6}$$

which is easier to work with because it only requires to compute expectations with respect to $p(w|\mathcal{D})$ and these can be easily approximated by Monte Carlo. The expression above also requires to compute the entropy of Bernoulli predictive distributions. Let $\mathcal{B}(z|p) = p^z(1-p)^{1-z}$ be a Bernoulli distribution on z with probability p. The entropy of $\mathcal{B}(z|p)$ can be easily obtained as

$$\mathrm{H}[\mathcal{B}(z|p)] = -p\log p - (1-p)\log(1-p) \equiv g(p)\,. \tag{7}$$

We could then sequentially construct $\mathcal{D}$ by optimising (6). However, this optimisation process would still be difficult, as it would require evaluating $\alpha(x_{1:T})$ exhaustively on all the elements of $\mathcal{X}$. To avoid this, we follow a greedy approach and construct our informative sequence in a sequential manner. In particular, at each time step $t = 1, \ldots, T$, we select x_t by optimising the mutual information between w and $Q^\star(x_{<t}, x_t)$, where $x_{<t}$ denotes here the prefix already selected at previous steps of the optimisation process. This mutual information quantity is denoted by $\alpha(x_t|x_{<t})$ and its expression is given by

$$\alpha(x_t|x_{<t}) = \mathrm{H}\{\mathbb{E}_{p(w|\mathcal{D})}[\mathcal{B}(z|\mathrm{y}(x_t|x_{<t}, w))]\} - \mathbb{E}_{p(w|\mathcal{D})}\{\mathrm{H}[\mathcal{B}(z|\mathrm{y}(x_t|x_{<t}, w))]\}\,, \tag{8}$$

where the binary variable z represents the model's predictions for $Q^\star(x_{<t}, x_t)$. The generation of an informative sequence can then be performed efficiently by sequentially optimizing (8), an operation that requires only $|\mathcal{C}| \times T$ evaluations of $\alpha(x_t|x_{<t})$.

To obtain an approximation to (8), we first approximate the posterior distribution $p(w|\mathcal{D})$ with $q(w)$ and then estimate the expectations in (8) by Monte Carlo using K samples drawn from $q(w)$. The resulting estimator is given by

$$\hat{\alpha}(x_t \mid x_{<t}) = g\left\{\frac{1}{K}\sum_{k=1}^{K} y(x_t|x_{<t}, w_k)\right\} - \frac{1}{K}\sum_{k=1}^{K} g[y(x_t|x_{<t}, w_k)]\,, \tag{9}$$

where $w_1, \ldots, w_K \sim q(w)$ and $g(\cdot)$ is defined in (7). The nonlinearity of $g(\cdot)$ means that our Monte Carlo approximation is biased, but still consistent. In practice, reasonable estimates can be obtained even for small K. In our experiments we use $K = 16$.

The iterative procedure just described is designed to produce a single informative sequence. However, in practice, we would like to generate a batch of informative and diverse sequences. The reason

for this is that, when training neural networks, processing a batch of data is computationally more efficient than processing individually multiple data points. To construct a batch with L informative sequences, we propose to repeat the previous iterative procedure L times. To introduce diversity in the batch-generation process, we "soften" the greedy maximisation operation at each step by injecting a small amount of noise in the evaluation of the objective function (Finkel et al., 2006). Besides introducing diversity, this can also lead to better overall solutions than those produced by the noiseless greedy approach (Cho, 2016). We introduce noise into the greedy selection process by sampling from

$$p(x_t|x_{<t}, \theta) = \frac{\exp\{\alpha(x_t|x_{<t})/\theta\}}{\sum_{x'_t \in \mathcal{C}} \exp\{\alpha(x'_t|x_{<t})/\theta\}} \tag{10}$$

for each $t = 1, \ldots, T$, which is a Boltzmann distribution with sampling temperature θ. By adjusting this temperature parameter, we can trade off the diversity of samples in the batch vs. their similarity.

3.2 Data augmentation

In some settings, such as the molecule domain we will consider later, we have databases of known-valid examples (e.g. collections of known drug-like molecules). Training directly on sequences which are valid has a danger of overfitting to only the subset of the space covered by the data, rather than the entire domain of $\mathcal{X}$; furthermore, if the model views primarily valid sequences during training then the batches of data will be quite non-uniform.

We address this by augmenting a database of valid sequences $x_{1:T}$ with additional, minimally-perturbed invalid sequences $x'_{1:T}$. This is constructed by setting each x'_t to be a symbol selected independently from $\mathcal{C}$ with probability γ, while remaining the original x_t with probability $1 - \gamma$. In expectation this changes γT entries in the sequence. We choose $\gamma = 0.05$, which generates synthetic data which, despite its high per-character overlap with original SMILES strings, empirically produces valid sequences less than 1% of samples.

4 Experiments

After training, we have learned a model $y(x_t|x_{<t}, w)$ which can be used to approximate the optimal $Q^\star(x_{<t}, x_t)$, and thus to estimate the potential for valid sub-sequences to have valid suffixes. To evaluate this we first consider how this Q-function approximation can be used to define optimal policies $\pi^\star$, exploring the tradeoff between accuracy rate when estimating $\tilde{v}(x_{1:t})$ and the coverage over the full space of valid prefixes in $\mathcal{X}$, in the context of generating mathematical expressions in the Python 3 programming language, and in generating valid molecules in the SMILES language.

4.1 Mathematical expressions

The value of sequential Bayesian active learning can be demonstrated in the context of Python 3 mathematical expressions. Here we base the expressions on the alphabet of numbers and symbols shown in table 1. Length 25 strings of these characters are run using the Python 3 `eval` function to evaluate their validity. This either returns the result of the calculation if valid or raises an exception. The expressions in this experiment are all of one fixed length.

Table 1: Python 3 expression alphabet

digits	operators	comparisons	brackets
`1234567890`	`+-*/%!`	`=<>`	`()`

The key in these experiments is that we do not assume the existence of a data set of positive examples that could be used in model training. To benchmark against the active learning approach, we train an identical network using data uniformly sampled from $\mathcal{X}$, which has 1.14×10^{33} total elements. Sampling uniformly for a very long duration allows us to estimate the total size of the valid subset $\mathcal{X}_+ \subset \mathcal{X}$ is approximately 3.17×10^{29}, meaning that drawing at random would only yield a valid sequence on the order of one every ten thousand samples.

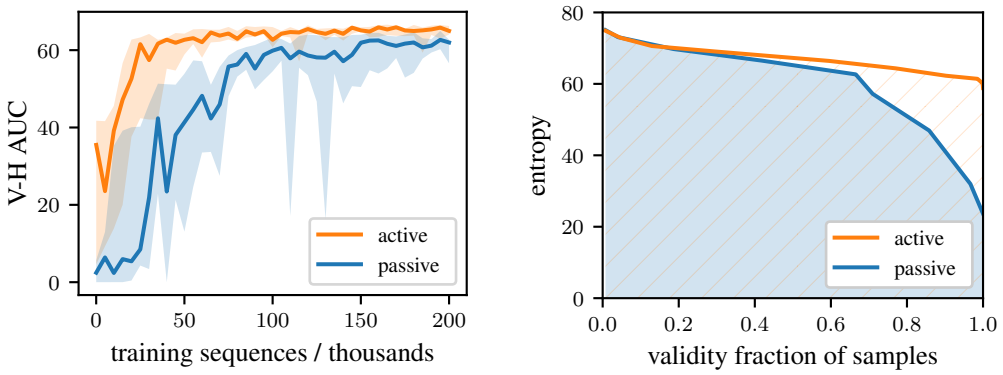

Figure 2: Experiments with length 25 Python expressions. (Left) Area under validity-entropy curve as training progresses. Active learning converges faster and reaches a higher maximum. (Right) Entropy versus validity for median active and median passive model after 200K training sequences. Both models have learnt to generate valid sequences with high entropy.

Table 2: Estimated lower bound of coverage N for Passive and Active models, defined as the size of the set of Python expressions the model places positive probability mass on. Evaluated on models trained until convergence, at $800,000$ training points (beyond scope of figure 2). The lower bound estimation method is detailed in Appendix B.

temperature τ	passive model		active model	
	validity	N	validity	N
0.100	0.850	9.7×10^{27}	0.841	8.2×10^{28}
0.025	0.969	2.9×10^{25}	0.995	4.3×10^{27}
0.005	1.000	1.1×10^{22}	1.000	1.3×10^{27}

Instead of defining a surrogate validation distribution to measure against, we measure the performance of the model in terms of its ability to generate diverse valid sequences. To do so requires defining a probabilistic model we can simulate from, as a function of our learned Q-function approximations $y(x_t|x_{<t}, w)$. A standard approach for this is a Boltzmann policy, i.e. a policy which samples next actions according to

$$\pi(x_t = c|x_{<t}, w, \tau) = \frac{\exp(y(c|x_{<t}, w)/\tau)}{\sum_{j \in \mathcal{C}} \exp(y(j|x_{<t}, w)/\tau)} \tag{11}$$

where τ is a temperature constant that governs the tradeoff between exploration and exploitation. Note that this is not the same as the Boltzmann distribution used as a proposal generation scheme during active learning, which was defined not on Q-function values but rather on the estimated mutual information. We obtain samples $\{x^{(1)}, \ldots, x^{(N)}\}_{\tau_i}$ for a range of temperatures τ_i and compute the validity fraction and entropy of each set of samples. These points now plot a curve of the tradeoff between validity and entropy that a given model provides. Without a preferred level of sequence validity, the area under this validity-entropy curve (V-H AUC) can be utilised as a metric of model quality.

The use of a Boltzmann policy allows us to tune the temperature parameter to identify policies which hit high levels of accuracy for any learned Q-function approximation. However, this is counterbalanced by a decrease in coverage, yielding false negatives. We will see that the Q-function approximation estimated by the active learning approach allows us to demand accuracy $> 99\%$ while still maintaining very high coverage over the space $\mathcal{X}$, as measured by the entropy of the distribution. Figure 2 shows the validity-entropy curve of the proposed Bayesian LSTM model trained on length 25 Python expressions, representing the tradeoff between policies at different sampling temperatures. To provide some context for these, we estimate an information theoretic lower bound for the fraction of the set $\mathcal{X}_+$ that our model is able to generate. This translates to upper bounding the false negative

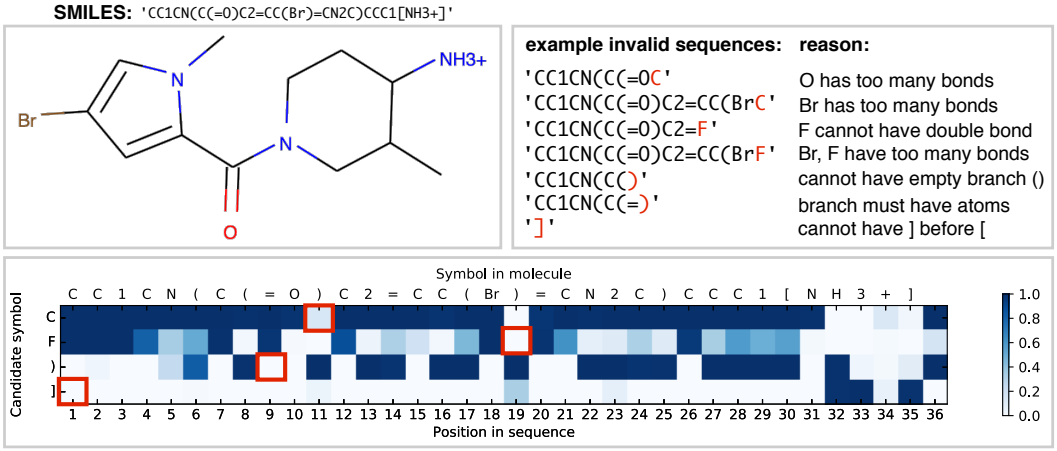

Figure 3: Predictions $y(x_t|x_{<t}, w)$ of the agent at each step t for the given (valid) test molecule at in the top left figure, shown over a subset of possible actions (selecting as next character `C`, `F`, `)`, or `]`) on the axis on the left. Each column shows which characters the trained agent believes are valid at each t, given the characters $x_{<t}$ preceding it. We see that the validity model has learned basic valence constraints: for example the oxygen atom `O` at position 10 can form at most 2 bonds, and since it is preceded by a double bond, the model knows that neither carbon `C` nor fluorine `F` can immediately follow it at position 11; we see the same after the bromine `Br` at position 18, which can only form a single bond. The model also correctly identifies that closing branch symbols `)` cannot immediately follow opening branches (after positions 6, 8, and 17), as well as that closing brackets `]` cannot occur until an open bracket has been followed by at least one atom (at positions 32–35). The full output heatmap for this example molecule is in Figure 4 in the appendix.

rate for our model. We also see Bayesian active learning converge faster per sample, with little additional computational cost; per 10,000 data points, training runtime is 31 seconds with the passive approach, and 38 seconds with active.

4.2 SMILES MOLECULES

One of the most common representations for molecules is as SMILES strings Weininger (1970). The top left of Figure 3 shows an example SMILES string and its corresponding molecule. A SMILES string is an ordering of atoms and bonds. It is attractive for many applications because it maps the graphical representation of a molecule to a sequential representation. This comes at the cost of intricate dependencies in SMILES strings that are dictated by chemical constraints. For instance, the atom Bromine (`Br`) can only bond with a single other atom. This means that it may only occur at the beginning or end of a SMILES string, or within a so-called 'branch' which are denoted with the characters `(`, `)`. The top right of Figure 3 shows more such examples: strings which are invalid SMILES because of a single error, shown in red. The full SMILES alphabet is shown in Table 3.

Table 3: SMILES alphabet

atoms/chirality	bonds/ringbonds	charges	branches/brackets
B CNO S P F I c n o s H Cl Br @	=#/\12345678	−+	() []

Whereas previous experiments focused on fixed length expressions, in order to test on real data sets we extend our framework to deal with variable length expressions. To so we introduce a terminal character ' ' to the alphabet. Because of the size and complexity of the space of possible molecules, at lengths $T > 20$ uniform sampling from the alphabet is unlikely to find a single valid molecule. Thus, we propose a new approach. We use a subset of the ZINC drug molecule dataset, used in Gómez-Bombarelli et al. (2016b), to train models of SMILES validity. The dataset consists of strings of length no more than $T \leq 120$, which implies a $|\mathcal{X}| = 5.65 \times 10^{189}$.

Dataset	passive model		active model	
	Accuracy (valid)	Accuracy (invalid)	Accuracy (valid)	Accuracy (invalid)
Zinc (test)	1.0	0.77	0.98	0.79
USPTO	0.96	0.77	0.86	0.70
Solubility	0.53	0.65	0.64	0.78

Table 4: Accuracy on known positive and negative examples of molecules. Negative examples are obtained using augmentation technique described in section 3.2.

We train two distinct models: (1) a data-augmented model which trains on minibatches of 10 ZINC sequences and 10 random sequences which are a mix of uniform random, and random generated by perturbing known-valid ZINC molecules; and (2) an active model, which includes 10 actively constructed sequences, 5 ZINC molecules, and 5 random which may be either uniform or perturbed ZINC molecules. These splits are chosen to obtain an approximately balanced number of positive and negative examples per batch; other parameters are specified in appendix A.

We test the predictions of the models on held-out data sequences, both from the ZINC dataset as well as from two completely external datasets (one a dataset for estimating molecule solubility, the other taken as a set of reaction outcomes from the USPTO 15k dataset). These are true held-out molecules, and represent challenging benchmarks for generalization. Recalling that a sequence is considered invalid if $\tilde{v}(x_{1:t}) = 0$ at any $t \leq T$, we take the prediction of each model for a molecule $x_{1:T}$ as $\prod_{t=1}^{T} \mathbb{I}\left[\mathrm{y}(x_t|x_{<t}, w) \geq 0.5\right]$, and compare this to its true label. Solubility contains many sequences that are not alike any seen in the Zinc training data. Unsurprisingly, the active model performs better in this domain. On the other two data sets, the augmentation technique is better at identifying valid molecules.

5 Discussion

By combining a Bayesian recurrent neural network with mutual-information-based active learning and a data augmentation scheme, the model we propose in this paper enables learning of valid sequences which would otherwise not be possible due to the vanishingly small number of valid sequences obtained when naïvely sampling from the data generating distribution.

These benefits are reflected in the experimental results: on a dataset of Python expressions with a length that allows enough valid examples to train a passive model, we show that the active learning approach converges much faster. For longer sequences the benefits are much greater: for the SMILES molecules, naively fitting to the data does not yield a model that is able to generate valid sequences or successfully generalize. Beyond providing practical improvements in drug discovery and program synthesis tasks, we hope this approach here can be adapted to learn validity estimators in new, novel domains.

References

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, (CoNLL)*, pp. 10–21, 2016.

Kyunghyun Cho. Noisy parallel approximate decoding for conditional recurrent language model. *arXiv preprint arXiv:1605.03835*, 2016.

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

Jenny Rose Finkel, Christopher D Manning, and Andrew Y Ng. Solving the problem of cascading errors: Approximate bayesian inference for linguistic annotation pipelines. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pp. 618–626. Association for Computational Linguistics, 2006.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 1019–1027, 2016.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2414–2423, 2016.

Rafael Gómez-Bombarelli, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, David Duvenaud, Dougal Maclaurin, Martin A Blood-Forsythe, Hyun Sik Chae, Markus Einzinger, Dong-Gwang Ha, Tony Wu, 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.

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

Gabriel L. Guimaraes, Benjamin Sanchez-Lengeling, Pedro Luis Cunha Farias, and Aln Aspuru-Guzik. Objective-reinforced generative adversarial networks (organ) for sequence generation models. In *arXiv:1705.10843*, 2017.

José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in neural information processing systems*, pp. 918–926, 2014.

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

Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

M. J. Kusner, B. Paige, and J. M. Hernández-Lobato. Grammar Variational Autoencoder. *International Conference on Machine Learning*, March 2017.

David JC MacKay. Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604, 1992.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv:1511.06434 [cs]*, November 2015.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. In *Proc. Edinburgh Math. SOC*, volume 17, pp. 1–14, 1970.

A APPENDIX: MODEL PARAMETERS AND IMPLEMENTATION DETAILS

Model hyperparameters. There are a handful of free parameters in both the LSTM model and the training procedure. We document the values used in all experiments in table 5. For both models we used a distributed representation of the characters with a size of 50 as input to the LSTM (i.e., an embedding layer of size 50). The "mixture weight" for samples chosen via randomization denotes the percentage which are drawn uniformly (i.e. 30%); the remainder are constructed via random perturbations of known-valid sequences.

parameter	passive	active
hidden size	512	512
dropout probability	0.0	0.2
batch size	20	20
temperature θ	–	1e-7
learning rate	1e-4	1e-4
weight decay	1e-8	1e-8
data augmentation γ	0.05	0.05
mixture weight for random samples	0.3	0.3

Table 5: Model parameters

B COVERAGE ESTIMATION

Ideally, we would like to check that the learned model $\mathrm{y}(x_t|x_{<t}, w)$ assigns positive probability to exactly those points which may lead to valid sequences, but for large discrete spaces this is impossible to compute or even accurately estimate. A simple check for accuracy could by to evaluate whether the model correctly identifies points as valid in a known, held-out validation or test set of real data, relative to randomly sampled sequences (which are nearly always invalid). However, if the validation set is too "similar" to the training data, even showing 100% accuracy in classifying these as valid may simply indicate having overfit to the training data: a discriminator which identifies data as similar to the training data needs to be accurate over a much smaller space than a discriminator which estimates validity over all of $\mathcal{X}$.

Instead, we propose to evaluate the trade-off between accuracy on a validation set, and an approximation to the size of the effect support of $\prod_t \mathrm{y}(x_t|x_{<t}, w)$ over $\mathcal{X}$. Let $\mathcal{X}_+$ denote the valid subset of $\mathcal{X}$. Suppose we estimate the fraction valid $f_+ = {}^{|\mathcal{X}_+|}/_{|\mathcal{X}|}$ by simple Monte Carlo, sampling uniformly from $\mathcal{X}$. We can then estimate $N_+ = |\mathcal{X}_+|$ by $f_+|\mathcal{X}|$, where $|\mathcal{X}| = C^T$, a known quantity. A uniform distribution over N_+ sequences would have an entropy of $\log N_+$. We denote the entropy of output from the model H. If our model were perfectly uniform over the sequences it can generate, it would then be capable of generating $N_{\text{model}} = e^H$. As our model at its optimum is extremely not uniform over elements $x \in \mathcal{X}$, this is very much a lower bound a coverage.

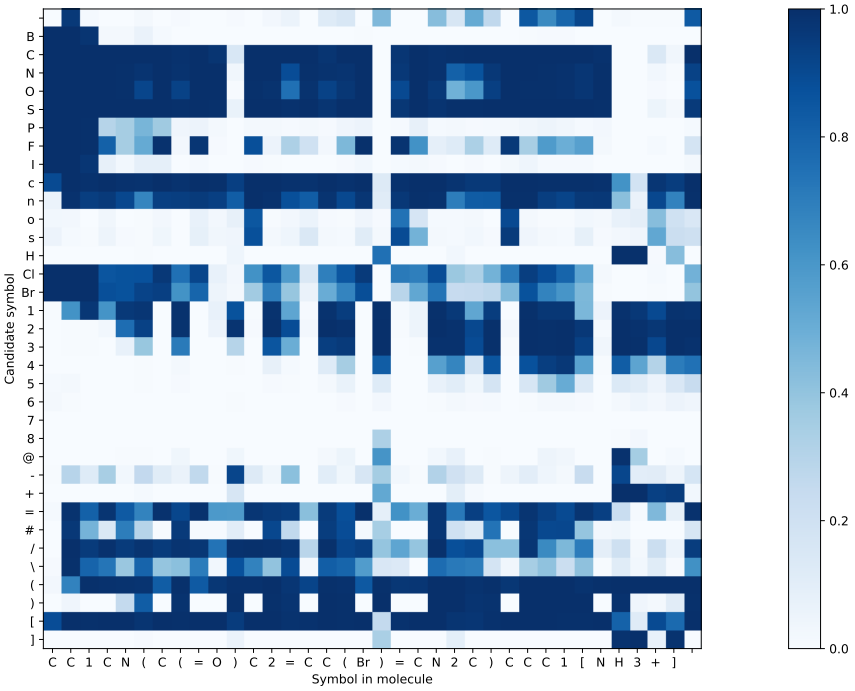

Figure 4: Full heatmap showing predictions $\mathrm{y}(x_t|x_{<t}, w)$ for the molecule in Figure 3.