# Neural Language Modeling for Molecule Generation

Sanjar Adilov
*Bioinformatics Laboratory*
*Romanovsky Institute of Mathematics*
Tashkent, Uzbekistan
s.adilov@mathinst.uz

***Abstract*—Generative neural networks have shown promising results in *de novo* drug design. Recent studies suggest that one of the efficient ways to produce novel molecules matching target properties is to model SMILES sequences using deep learning in a way similar to language modeling in natural language processing. In this paper, we present a survey of various machine learning methods for SMILES-based language modeling and propose our benchmarking results on a standardized subset of ChEMBL database.**

***Index Terms*—drug design, language modeling, deep learning, recurrent neural networks**

## I. Introduction

In *de novo* molecular design, we seek to produce novel molecule libraries that meet required property profiles. Our aim is to create generative statistical models for molecular data that can capture the distributions of molecular compounds and generate a new set of valid compounds with the desired property requirements. Some notable advances in complete de novo drug design are established by Segler et al. [1] and Gupta et al. [2]; they introduce *generative recurrent networks*, which are trained on a large and diverse data set to recognize general features of molecules and fine-tuned on a smaller focused set active towards the desired target. Brown et al. [3] and Polykovsky et al. [4] attempt to compose standardized benchmarking frameworks with various generative models as well as distribution and property-based evaluation metrics.

The choice of a particular generative model depends on several aspects of the given task and materials. Considering data, the most common representation of a molecule is *simplified molecular-input line-entry system* (*SMILES*) notation [5], which comprises ASCII characters structured by a special language rule. This specification immediately suggests the possibility of building SMILES-based language models, in which a SMILES character can be treated as a single *token*, a sequence of one or more tokens as a (meaningful) *text* or *phrase*, and the whole text data set as a language *corpus*. Inspired by such natural language models as $n$-grams and recurrent neural networks [6], we are free to simulate our own language framework capable of capturing SMILES fragments and generating novel sequences.

In this work, we attempt to provide a comprehensive outline of SMILES-based neural molecule generation problem: formulation of a language model, preprocessing of sequence data, generation strategies, and evaluation of models and generated data. We first formulate a general framework of estimating the joint probability of a sequence, and given a SMILES data corpus, discuss data preparation methods suited for this framework. Next, we present a neural network architecture for language modeling, focusing particularly on recurrent neural networks. Using trained neural networks, we describe how to generate novel data and assess their validity, diversity, and other molecular properties. Finally, we test the presented SMILES language framework on a large corpus and discuss the obtained results as well as future perspectives regarding new language models. In our repository MoleculeGen-ML [a], we introduce a Python package for experimenting with the presented methods and reproducing the benchmarking results. Neural networks in the package are implemented using MXNet backend [7] and follow its Gluon API [8]; evaluation methods use RDKit package [9].

## II. General Framework

Given a corpus of texts written in some natural or formal language, we seek to learn the grammar and semantics of this language by analyzing the texts via probabilistic modeling. Texts are composed of basic units called *tokens*, each of which is derived from some *vocabulary*, a set of available tokens. Suppose that we have a text $\mathbf{s}$. To begin our analysis, we first need to *tokenize* the text, i.e. establish the set of rules to divide $\mathbf{s}$ into tokens $s_1, s_2, \ldots, s_T$ (e.g. partition an English text to obtain words and punctuation marks). Depending on tokenization method, our vocabulary $\mathcal{V}$ will comprise unique tokens from processed $\mathbf{s}$.

Now, we have the text sequence $\boldsymbol{s} = (s_1, s_2, \ldots, s_T)$ composed of $T$ ordered tokens $s_t$ at *time steps* $t$. The goal of a *language model* is to estimate the joint probability of the entire sequence $\boldsymbol{s}$:

$$\begin{aligned} P(\mathbf{s}) &= P(s_1) \cdot P(s_2|s_1) \cdots P(s_T|s_1, s_2, \ldots, s_{T-1}) \\ &= \prod_{t=1}^{T} P(s_t|s_1, \ldots, s_{t-1}) = \prod_{t=1}^{T} P(s_t|s_{j<t}). \end{aligned}$$

So, having observed previous $t-1$ tokens, we would like to predict the next token $s_t$, $1 \leq t \leq T$, via conditional probability, *from left to right*. How should we calculate the probabilities? First, note that lengths of different sequences may vary, and we can rationally assume that it is sufficient to observe only tokens $s_{t-\tau}, \ldots, s_{t-1}$ in some *time span* $\tau < t$

[a] http://www.github.com/sanjaradylov/moleculegen-ml

to predict $s_t$. This way, probabilities can be estimated by counting all occurrences of tokens divided by the vocabulary dimension and additionally using some smoothing techniques. This is how classic $n$-*gram language modeling* works:

$$P(s_t|s_1,\ldots,s_{t-1}) = P(s_t|s_{t-\tau},\ldots,s_{t-1}),$$

$$\hat{P}(s_t|s_1,\ldots,s_{t-1}) = \frac{\#(s_1,\ldots,s_{t-1},s_t)+\epsilon}{\#(s_1,\ldots,s_{t-1})+\epsilon|\mathcal{V}|}.$$

On the other hand, we could model a time-dependent summary $h_t \in \mathcal{R}$ of the past observations so that $s_t \sim P(s_t|h_{t-1})$ and for some mapping $f : \mathcal{V} \times \mathcal{R} \to \mathcal{R}$, $h_t = f(s_t, h_{t-1})$. This strategy is called *latent autoregressive modeling*. Contrary to $n$-grams, it does not require tuning the time span hyperparameter and storing sparse parameters (probabilities), which will likely capture only short contexts and overfit the training dataset. Recurrent neural networks (RNNs) can be viewed as latent autoregressive models. In Section IV-B, we describe SMILESRNN architecture, which was chosen to be the baseline generative language model.

## III. Working with Data

### A. Dataset Collection

To create a comprehensive generative language model, we need a large and diverse set of SMILES data. *ChEMBL* database [10] comprises millions of molecular compounds with their measured biological activity. One can freely download the database and collect a desirable set of SMILES strings of synthesized molecules. Note that for more convenient modeling, canonicalization (determining one special SMILES representation among all valid ones) and subsequent removal of duplicates may be required. Also, lengths of SMILES strings can be much longer than average suggesting that large outlier molecules can reasonably be filtered out. These and other preprocessing techniques, including charge neutralization and removal of salts and molecules with prohibited subcompounds, were performed by Brown et al. [1], which led to a dataset of almost 1.3 million SMILES strings from ChEMBL 24. We will train and evaluate our language framework on this dataset.

### B. Tokenization

Recall that SMILES is a line notation composed of an encoded series of characters representing molecule constituents: atoms (**B**, **Cl**, etc.), bonds (**-**, **=**, **#**, **:**), branches (**(**, **)**), together with substructure and property specifications. For example, the SMILES string for Acetate ($C_2H_3O_2^-$) is **CC(=O)[O-]**. So how should we tokenize such texts? One obvious way is a *character-level* tokenization, i.e. treating every character as a token. However, it might be counterintuitive to break up multiple-character entities having inseparable encodings (e.g. **Br**, **Sr**). To develop a more coherent strategy, we will try following at least basic SMILES language rules. First, our algorithm will match both single- and multi-character symbols by comparison with the set of available atomic and non-atomic symbols. Note that numbers greater than **9** follow **%** character, so **%10** will become a separate token. Second, as atoms in aromatic rings are specified by lower case symbols, they will be treated as separate tokens (e.g. **n**, **[se]**). Finally, we will provide an option to capture special aggregate symbols such as hydrogen specifications (e.g. **[N+]**).

With all these conventions, our Acetate string can be tokenized as (**C**, **C**, **(**, **=**, **O**, **)**, **[O-]**) and the vocabulary will consist of {**C**, **(**, **=**, **O**, **)**, **[O-]**}. For our setup, the proposed tokenization method is applicable, however, for some other tasks and different data collection, one may have to set up additional refinements to match more complex SMILES specifications and conventions (e.g. chirality, isotopism, etc.).

### C. Minibatch Sampling

SMILES entries can be arbitrarily long. As we train a generative neural network, where we invoke *minibatch stochastic gradient descend* for parameter optimization and process sequences of fixed length, we need to decide on how to arrange and slice SMILES strings into subsequences of the chosen length to sample minibatches.

But before we begin, there are three special tokens that we should take into consideration during both training and generation processes. First, we need a special *beginning-of-SMILES* token **{**, from which a model learns how to sample the very first token. Of course the model will be able to capture any subsequence as the starting fragment of a sequence being generated, but we might want to commence generation arbitrarily. Therefore, every SMILES string will be prepended by **{** prior to training. Next, our model should somehow decide when to terminate sequence generation. One option is to define the *maximum number of tokens* hyperparameter. However, this will force us to guess a "good number" for every novel sequence. Instead, we will append a special *end-of-SMILES* token **}** to every SMILES string prior to training so that the model will also learn and sample **}** indicating the termination of the process. The maximum number of tokens will be a supplementary option if we wish to force the creation of medium-length sequences. Finally, some SMILES strings may be shorter than the chosen time span, so we will have to lengthen them with a special *padding* token **_** that will be ignored during training and not likely be sampled during generation.

Now, let us introduce minibatch sampling strategies. Let $\ell$ be a minibatch size and $\tau$ time span. Let $\boldsymbol{S} = (\boldsymbol{s}_1,\ldots,\boldsymbol{s}_L)$ be a training corpus with SMILES strings $\boldsymbol{s}_i$, $1 \le i \le L$. Suppose we have already tokenized the data and obtained a vocabulary $\mathcal{V}$, which also includes tokens **{**, **}**, and **_**. Our goal is to produce an input subsequence $\boldsymbol{X} \in \mathcal{V}^{\ell\times\tau}$ and target/output subsequence $\boldsymbol{Y} \in \mathcal{V}^{\ell\times\tau}$ (the input shifted by one token).

*1) Column Sampling:* Collect $\ell$ SMILES sequences, append padding tokens up to the maximum length in the collection, divide them into minibatches of shape $(\ell,\tau)$ column-wise, and generate them successively (Figure 1).

*2) Consecutive Sampling:* Select a SMILES sequence from $\mathbf{S}$, slice it into subsequences of length $\tau$, and append padding tokens to the last one if necessary. Repeat this strategy until $\ell$ subsequences are sampled.

*3) Random Sampling:* Let $M$ be the maximum length in $\boldsymbol{S}$. Let $m$, $0 \leq m \leq M - \tau$, be the largest index from which a subsequence will be picked. This is another hyperparameter and we call it the *maximum offset*. Sample an integer $k$, where $m \leq k \leq M - \tau - m$, select a sequence $\boldsymbol{s}$ from $\boldsymbol{S}$, and sample a subsequence $\boldsymbol{s}[k..k+\tau]$ padding the necessary number of tokens. Repeat until $\ell$ subsequences are sampled.

Incidentally, the maximum offset can be introduced for the first two strategies as well. To prevent sampling of an empty subsequence, we could choose it to be the minimum length in $\boldsymbol{S}$ minus 2. Random sampling may also create empty subsequences, and one way to guarantee nonempty entries is to sample the required number of minibatches using consecutive sampling, integrate them into one batch, and randomly pick $\ell$ subsequences with replacement.

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| { | N | C | C | = | O | } | _ | _ | _ | _ | _ | _ | _ | _ |
| { | N | c | 1 | c | c | ( | O | ) | n | c | c | 1 | F | } |
| { | C | C | C | ( | C | ) | ( | C | ) | Br | } | _ | _ | _ |
| { | C | n | 1 | [nH] | n | n | c | 1 | = | S | } | _ | _ | _ |

Fig. 1. Three input minibatches of size $4 \times 5$ derived from column sampling.

## IV. Methods

### A. SMILESLM

An abstract neural network architecture for training SMILES language models comprising three stacked blocks (Figure 2):

- *embedding* $\mathrm{E} : \mathcal{V}^{\ell \times \tau} \to \mathbb{R}^{\ell \times \tau \times d}$;
- *encoder* $\Phi : \mathbb{R}^{\ell \times \tau \times d} \to \mathbb{R}^{\ell \times \tau \times h}$;
- *output* $\Psi : \mathbb{R}^{\ell \times \tau \times h} \to \mathbb{R}^{\ell \times \tau \times |\mathcal{V}|}$.

*1) Embedding:* Transformation of token sequences into expressive feature representation of the predefined dimension $d$. Can be *one-hot* encoding (creating token feature mask of length $d = |\mathcal{V}|$), parameterized hidden layer, or fixed/parameterized positional encoding [11]. Optionally followed by a *dropout* layer [12] that during training stage, randomly zeros out entire feature column (i.e. "removes" chosen feature), entire token representation, or both.

*2) Encoder:* Neural network with an optional hidden state/memory mechanism that captures sequential information from obtained embedding features. Parameterized latent autoregressive models are an obvious example.

*3) Output:* Decoding encoder's sequence representations into probability distributions for the subsequent tokens. Can be a linear projection followed by $\mathrm{softmax}$, $\mathrm{gumbel\text{-}softmax}$ [13], $\mathrm{sparsemax}$ [14], or analogous probability transformations. Note that using learnable embedding allows *parameter sharing* (or *tying weights*) [15] with the output block as it can similarly be viewed as the output token embedding. It reduces the model size and creates an effect of optimizing the regularized objective taking into account the similarity between the target token and other tokens from the vocabulary.

SMILESLM predicts the probabilities of the next token given the previous context. To train the model, we need an objective that maximizes predicted probabilities to the target tokens. It is equivalent to minimizing the *cross-entropy loss* between one-hot encoded target distributions and predicted probabilities of tokens: for a target minibatch $\boldsymbol{Y} \in \mathcal{V}^{\ell \times \tau}$ and output probabilities $\mathbf{P} \in [0,1]^{\ell \times \tau \times |\mathcal{V}|}$,

$$
\begin{aligned}
Q(\boldsymbol{Y}, \mathbf{P}) &= \frac{1}{\ell\tau} \sum_{i=1}^{\ell} \sum_{t=1}^{\tau} H\big(\text{one-hot}(y_{i,t}), \boldsymbol{p}_{i,t}\big) \\
&= -\frac{1}{\ell\tau} \sum_{i=1}^{\ell} \sum_{t=1}^{\tau} \sum_{k=1}^{|\mathcal{V}|} \text{one-hot}(y_{i,t})_k \cdot \log p_{i,t,k}
\end{aligned}
$$

Note that one-hot encoding a target token means creating the weight vector of length $|\mathcal{V}|$ assigning 1 for the target label and 0 for the rest. Yet we might choose the label weight of the token to be inversely proportional to its occurrences in the batch. Also, recall that padding token _ should be ignored, so its label weight will be assigned to 0.

We train SMILESLM with minibatch stochastic gradient descend for parameter updates and backpropagation [6] for gradient calculations. Since we process long sequences, we try to prevent gradient explosion by applying gradient clipping. To reduce overfitting, we employ dropout between hidden layers in the encoder, set up a learning rate scheduler, and monitor the progress on the generated data with early stopping.

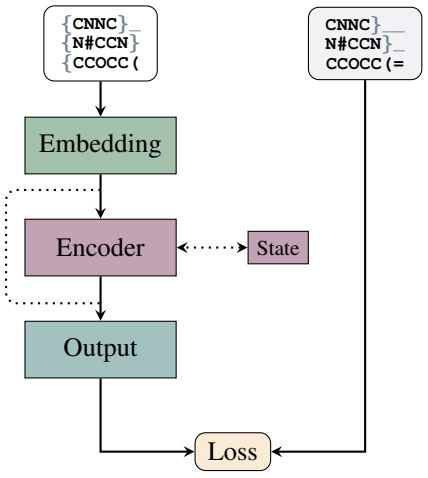


Fig. 2. SMILESLM neural network architecture.

### B. SMILESRNN

Recurrent neural networks are neural networks with latent variables called *hidden states*. As mentioned in Section II, latent variable models are more preferable compared to $n$-gram models at least because the number of parameters of the latter is $O(|\mathcal{V}|^n)$ and it grows exponentially with $n$. Using RNNs as an encoder block in SMILESLM gives SMILESRNN model. Assume that $\boldsymbol{X}_t \in \mathbb{R}^{\ell \times d}$ is a minibatch of embedding features at time step $t \leq \tau$, $\boldsymbol{H}_t \in \mathbb{R}^{\ell \times h}$ a hidden state at $t$, and $\boldsymbol{O}_t \in \mathbb{R}^{\ell \times |\mathcal{V}|}$ an output at $t$. The hidden state at $t$ depends on

the current inputs and the previous state, while the output at $t$ on the current hidden state:

$$\boldsymbol{H}_t = \phi(\boldsymbol{X}_t\boldsymbol{W}_{xh} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hh} + \boldsymbol{b}_h),$$
$$\boldsymbol{O}_t = \psi(\boldsymbol{H}_t\boldsymbol{W}_{ho} + \boldsymbol{b}_o).$$

$\boldsymbol{W}_{xh} \in \mathbb{R}^{d\times h}$, $\boldsymbol{W}_{hh} \in \mathbb{R}^{h\times h}$, $\boldsymbol{W}_{ho} \in \mathbb{R}^{h\times|\mathcal{V}|}$, $\boldsymbol{b}_h \in \mathbb{R}^h$, and $\boldsymbol{b}_o \in \mathbb{R}^{|\mathcal{V}|}$ are learnable parameters; $\phi$ is an element-wise activation function (typically $\tanh$); $\psi$ is another activation function to get output distributions (e.g. $\mathrm{softmax}$).

This is how vanilla RNNs are constructed. At each time step during forward propagation, they summarize the previous context in the state variable and perform recurrent computation of the next state to determine the next output distribution. Backward propagation requires unfolding the computational graph of depth $\tau$ (*backpropagation through time* or *BPTT* [16]). The longer the input sequences are, the more chances to experience gradient vanishing or exploding, which results in numerical instability. Typically, it means that earlier sequence fragments in the context are more or less relevant for future observations. To address such peculiarities, various gating mechanisms such as in *long short-term memory* (*LSTM*) [17] and *gated recurrent units* (*GRU*) [18] were proposed. For example, GRUs attempt to capture information importance, irrelevance, and logical break by incorporating *reset gates* $\boldsymbol{R}_t \in (0,1)^{\ell\times h}$ for short-term dependencies and *update gates* $\boldsymbol{U}_t \in (0,1)^{\ell\times h}$ for long-term dependencies:

$$\boldsymbol{R}_t = \sigma(\boldsymbol{X}_t\boldsymbol{W}_{xr} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hr} + \boldsymbol{b}_r),$$
$$\boldsymbol{C}_t = \phi\big(\boldsymbol{X}_t\boldsymbol{W}_{xh} + (\boldsymbol{R}_t \odot \boldsymbol{H}_{t-1})\boldsymbol{W}_{hh} + \boldsymbol{b}_h\big),$$
$$\boldsymbol{U}_t = \sigma(\boldsymbol{X}_t\boldsymbol{W}_{xu} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hu} + \boldsymbol{b}_u),$$
$$\boldsymbol{H}_t = \boldsymbol{U}_t \odot \boldsymbol{H}_{t-1} + (1 - \boldsymbol{U}_t) \odot \boldsymbol{C}_t.$$

$\boldsymbol{C}_t$ is called a *candidate hidden state*. Reasonably, hidden states at $t = 0$ are initialized with zeros so that gates and states at $t = 1$ process sequences with no past information. However, noisy initialization may be practical as well since the network processing subsequences of arbitrary length will not adapt to the zero state. Apart from initialization, we should also decide on how to reset states every training iteration. Intuitively for column sampling, we will keep the previous state for a new minibatch until we encounter the minibatch with the final column, i.e. the subsequences ending with end-of-SMILES or padding tokens. For consecutive sampling, we might want to retain or even train the state throughout an epoch. Random sampling will likely fit with noisy state reinitialization. Lastly, depending on the preferred sampling technique, we will detach hidden states from the computational graph (*truncated BPTT* [16]).

Big and complex datasets might require more flexibility than what single-layer RNNs offer. For a better sequence representation, we might as well stack multiple recurrent layers hypothesizing that higher layers capture longer-term dynamics and lower layers shorter-term. For a $K$-layer RNN, the $k^{\text{th}}$ layer at time step $t$ is $\boldsymbol{H}_t^{(k)}$ and it depends on the input $\boldsymbol{H}_t^{(k-1)}$ and the hidden state $\boldsymbol{H}_{t-1}^{(k)}$; $1 \le k \le K$, $\boldsymbol{H}_t^{(0)} = \boldsymbol{X}_t$. To tackle overfitting, we can incorporate dropout between layers, either regular [12] or *variational* [19], which creates input, hidden, and output masks and repetitively applies them at each time step.

### C. Generation Strategies

To generate a diverse set of coherent SMILES strings, SMILESLM will successively sample tokens from the predicted probability distributions based on the output block $\Psi$. If we wish to produce strings from the very beginning, without any context constraint, we specify beginning-of-SMILES token **{** as the first token. The process will be terminated upon predicting end-of-SMILES token **}** or reaching the length constraint.

If a model supports memory mechanisms, subsequent predictions will also be based on the previous ones. Thus, SMILESRNN will pass the currently predicted token back to the model to predict the next one (*teacher forcing*, Figure 3). Another important remark is that having received a SMILES fragment instead of **{**, the model successively updates its memory without making predictions (*warm-up* period), then proceeds generating the remaining fragment.

*1) Sampling with Temperature:* Toggling the intensity of probabilities by perturbing logits with *temperature* (sensitivity) hyperparameter $\delta > 0$. For the penultimate layer output of $i^{\text{th}}$ sequence at time step $t$ $\boldsymbol{z} = [z_1, \ldots, z_{|\mathcal{V}|}]^T$, $\mathrm{softmax}$ with temperature is defined as:

$$\mathrm{softmax}(\boldsymbol{z})_k = \frac{\exp(z_k/\delta)}{\sum_{j=1}^{|\mathcal{V}|}\exp(z_j/\delta)} \quad \text{for } k = 1, \ldots, |\mathcal{V}|.$$

As $\delta \to 0$, it approximates $\mathrm{argmax}$ (*greedy search*). Overall, there is a diversity-coherence tradeoff.

*2) Top-k Sampling:* Picking from the $k \ge 1$ most probable tokens $\mathcal{V}^{(k)} \subset \mathcal{V}$, rescaling their probabilities, and sampling based on the derived distribution. It is more optimal than tuning $\delta$ if top-$k$ tokens cover the large portion of the probability mass or all have high probabilities.

*3) Top-p (Nucleus) Sampling:* [20] Sampling from the smallest vocabulary subset $\mathcal{V}^{(p)} \subset \mathcal{V}$ such that it takes up the most probable tokens whose cumulative probability mass exceeds $p$, $0 < p < 1$. Fixed $k$ can be suboptimal across different distributions, while a top-$p\%$ subset is dynamic and for high values of $p$ covers the majority of the probability mass.

### D. Evaluation Metrics

The following metrics assess the ability of generative models to produce a diverse and acceptable set of molecules similar to a reference/training set.

*1) Perplexity:* Harmonic mean of the number of token choices during training:

$$\mathrm{PPL}(\boldsymbol{Y}, \mathbf{P}) = \frac{1}{\ell}\sum_{i=1}^{\ell}\exp\left(\frac{1}{\tau}\sum_{t=1}^{\tau} H\big(\text{one-hot}(y_{i,t}), \boldsymbol{p}_{i,t}\big)\right).$$

*2) Validity:* The rate of the SMILES strings corresponding to realistic molecules.

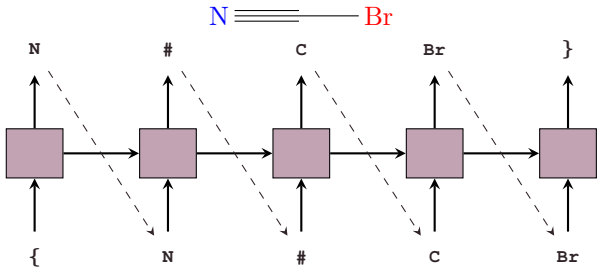


Fig. 3. SMILES sequence generation with teacher forcing using SMILESRNN.

*3) Uniqueness:* The rate of unique SMILES strings (regardless of their validity).

*4) Novelty:* The rate of the SMILES strings not presented in a reference set.

*5) Rate of Acceptable Compounds:* The rate of valid, unique, and novel SMILES strings.

*6) Internal Diversity:* [21] Average pairwise dissimilarity measure based on *Tanimoto distance* between *Morgan fingerprints* [22] (*ECFP*) of two molecules in a molecule set $\boldsymbol{M}$:

$$\mathrm{IntDiv}(\boldsymbol{M}) = 1 - \frac{1}{|\boldsymbol{M}|^2} \sum_{\boldsymbol{m},\boldsymbol{m}' \in \boldsymbol{M}} T(\boldsymbol{m}, \boldsymbol{m}'),$$

$$T(\boldsymbol{m}, \boldsymbol{m}') = \frac{|\boldsymbol{m} \cap \boldsymbol{m}'|}{|\boldsymbol{m} \cup \boldsymbol{m}'|}.$$

*7) Nearest Neighbor Similarity:* [4] Average Tanimoto similarity between Morgan fingerprints of a molecule from $\boldsymbol{M}$ and its nearest neighbor in $\boldsymbol{R}$:

$$\mathrm{SNN}(\boldsymbol{M}, \boldsymbol{R}) = \frac{1}{|\boldsymbol{M}|} \sum_{\boldsymbol{m} \in \boldsymbol{M}} \max_{\boldsymbol{r} \in \boldsymbol{R}} T(\boldsymbol{m}, \boldsymbol{r}).$$

*8) KL Divergence:* [3] Average descriptor similarity between $\boldsymbol{M}$ and $\boldsymbol{R}$ based on the Kullback-Leibler divergence:

$$D_{KL}(\boldsymbol{p} \parallel \boldsymbol{q}) = \sum_i p_i \log \frac{p_i}{q_i},$$

$$\mathrm{KL}(\boldsymbol{M}, \boldsymbol{R}) = \frac{1}{|\mathfrak{D}|} \sum_{\mathfrak{d} \in \mathfrak{D}} \exp\Big( - D_{KL}\big(\mathfrak{d}(\boldsymbol{M}) \parallel \mathfrak{d}(\boldsymbol{R})\big)\Big),$$

where $\mathfrak{D}$ is the set of target descriptors (e.g. *physicochemical* [3]) and $\mathfrak{d}$ is mapping from a molecule set into a descriptor distribution. For continuous descriptors, distributions are calculated via kernel density estimation, and for discrete descriptors via histograms.

## V. Experiments and Discussion

Now let us present our language modeling pipeline, from which we obtain benchmarking results. As we experiment on the standardized dataset from Brown et al. (see Section III-A), we do not perform any additional preprocessing except for removal of relatively short compounds (SMILES strings of length less than 5). Next we tokenize our data as discussed in Section III-B without capturing aggregate subcompounds. We noticed that simpler tokenization gives higher overall scores for every model. With the restriction on the minimum number of occurrences equalling 20, our vocabulary (including special tokens) has dimension 42. Having the tokenized corpus, we choose a minibatch sampling method. Our experiments suggest that random sampling generalizes better with recurrent networks trained on bigger datasets. We apply randomized consecutive sampling: set the maximum offset to 2, derive a collection of subsequences from consecutive sampling, and sample 80% of them with replacement.

The minibatch size and time span are set to 128 and 64, respectively. The choice of a time span has more effect on final results and we suggest experimenting on values between 64 and 80. The batch sizes between 64 and 128 with the corresponding adjustments on learning rates give almost identical results. The initial and final learning rates are set to 0.001 and 0.0001, respectively, with intermediate updates on every iteration of minibatch SGD (we use the *Adam* optimizer [23]) according to the *cosine learning scheduler*. The gradient clipping radius is set to 10, although with shorter time spans, gradient exploding occurs rarely.

We employ a parameterized embedding layer with output dimension 32 and a subsequent dropout on features of rate 0.4; a two-layer LSTM with 256 hidden units each and a dropout of rate 0.6 between recurrent layers; a linear layer with no parameter sharing. Having an additional, penultimate layer with the number of units equalling to the embedding dimension allows parameter sharing, and from our experience, it works better for larger vocabularies. All feed-forward layers are initialized with the *Xavier* algorithm [24] and recurrent layers with orthogonal initialization. On every iteration, the hidden states are detached from the computational graph and reinitialized to zeros.

On every epoch, we generate 1,000 sequences with top-80% gumbel-softmax activation, with the maximum length restriction of 100 corresponding to the training dataset statistics. Experimentally, fixed $p$ in top-$p$ sampling is almost always optimal on every iteration and epoch, while temperature or top-$k$ sampling require constant adjustment of their respective hyperparameters. Generated sets are evaluated using metrics from Section IV-D. The radius and length of Morgan fingerprints are set to 2 and 1024, respectively. Validity and Uniqueness are assessed on the first 1,000 strings; to get Novelty scores, invalid and repetitive strings are removed and newly generated until a supplemented set is unique and valid; IntDiv, SNN, and KL are correspondingly evaluated on a generated set of 1,000 novel strings. Optionally, RAC or loss values are monitored for early stopping. We present the results on the 12$^{\text{th}}$ epoch (see Table I), although epochs 5-7 already demonstrate reasonable Validity, Novelty, and IntDiv scores.

Figure 4 shows the distributions of quantitative estimation of drug-likeness [25] and Figure 5 the nonmetric *multidimensional scaling* of physicochemical descriptors of the training and generated sets. Along with the metric scores, they give evidence of property and fragment closeness of both sets. Figure 6 visualizes six randomly chosen molecules from the generated set.

TABLE I
SMILESRNN evaluation results averaged over 3 tests with different model initialization.

| PPL | Valid. | Uniq. | IntDiv[a] | Novel. | SNN | KL |
|---|---|---|---|---|---|---|
| 1.919 (±0.008) | 0.973 (±0.004) | 1.000 (±0.000) | 0.846 (±0.002) | 0.969 (±0.002) | 0.619 (±0.004) | 0.965 (±0.009) |

[a] IntDiv of the training set is 0.877.

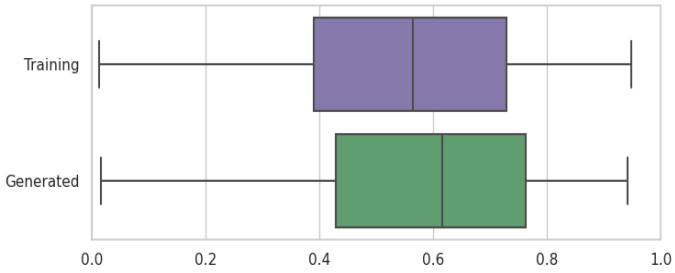


Fig. 4. Distributions of QED.

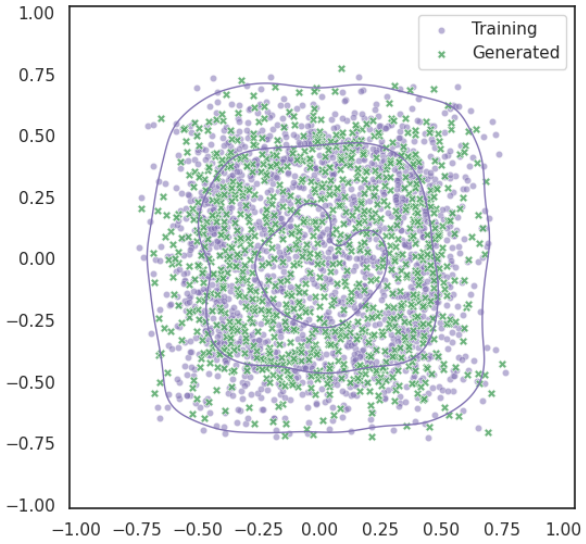


Fig. 5. MDS of Physicochemical Descriptors.

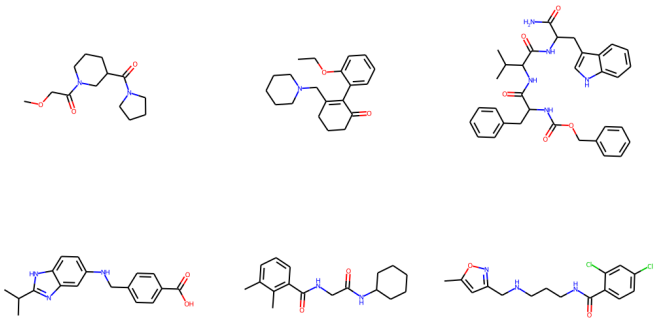

Fig. 6. Examples of generated molecules.

## VI. Notes on Other Encoders

Although multi-layer LSTM and GRUs demonstrate appropriate and promising performance, one can also build an encoder block of SMILESLM using *convolutional layers* and *attention mechanisms*. Depending on data, task, and network architecture, one might benefit from parallelization and interpretability of such encoders. The following two subsections describe CNN and Transformer encoders, which have found significant success in NLP tasks. However, based on our experience, they are inferior in terms of validity and capturing of long-term SMILES fragments. As both models need only constant minimum number of sequential operations and have shorter maximum path lengths, this seems counterintuitive; still, we hypothesize that the main reason is the absence of an explicit state/memory mechanism. We suggest experimenting with more refined models like in [26] or [27] and comparing their performances with our baseline or the results from [3].

### A. SMILESCNN

A CNN encoder block mainly consists of a stack of identical causal or dilated convolutional layers. Each convolution has stride 1 and prior to processing prepends $k-1$ padding tokens to inputs, where $k$ is a kernel size, so that outputs of convolutions have dimensions equal to the inputs'. This way we save positional information (and therefore, pooling is also redundant). Kernels can be interpreted as fragment detectors. The maximum path length is $O(\tau/k)$ or $O(\log_k \tau)$, depending on the type of convolutions. The computational complexity of convolutions is $O(k\tau d^2)$, as opposed to $O(\tau d^2)$ of RNNs, but the sequential operations can be parallelized. Optionally, for deeper models, we employ *residual* [28] or *highway* [29] connection between the outputs of an embedding block and the last convolutional layer.

### B. SMILESTransformer

Transformer encoder block is identical to the decoder part of the original model [11]. It consists of a stack of identical layers incorporating masked multi-head self-attention pooling [11] with residual connection, layer normalization [30], and position-wise feed-forward network with residual connection. Self-attention sublayer preserves autoregressive property by masking out future tokens and independently tries to attend different fragments with multiple heads. It is then followed by a normalization across the embedding features. The obtained representations at all the time steps are processed using the same feed-forward network and then also followed by layer normalization. The computational complexity is $O(\tau^2 d)$ with constant minimum number of sequential operations. Optionally, dropouts are applied to the attention weights and/or the output of the multi-head attention.

## VII. Conclusion

In this paper, we provided a suite of methods for SMILES-based language modeling. The presented data processing and neural network modeling techniques are similar to those in natural language processing, and during evaluation, demonstrate the ability to generate a diverse set of novel molecules matching the properties of the reference dataset. In fact, these methods can also be adapted to other tasks such as target classification and transfer learning. We hope that this survey will be a useful guide in developing further SMILES-based molecule generation frameworks.

## References


[1] M. Segler et al. (2018). *Generating Focused Molecule Libraries for Drug Discovery with Recurrent Neural Networks. ACS* Cent. Sci., 4, 120-131.

[2] A. Gupta et al. (2018). *Generative Recurrent Networks for de Novo Drug Design*. Mol. Inf. 37, 1700111.

[3] N. Brown et al. (2019). *Guacamol: benchmarking models for de novo molecular design*. J. Chem. Inf. Model. 59, 1096–1108.

[4] D. Polykovskiy et al. (2020) *Molecular sets (moses): a benchmarking platform for molecular generation models*. Front Pharmacol 11:58.

[5] D. Weininger. (1988). *SMILES, a Chemical Language and Information System. Introduction to Methodology and Encoding Rules*. J.Chem. Inf. Comput. Sci. 28, 31.

[6] D. Rumelhart et al. (1988). *Learning representations by back-propagating errors*. Cognitive modeling, 5(3), 1.

[7] T. Chen et al. (2015). *MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems*. arXiv preprint arXiv:1512.01274.

[8] A. Zhang et al. (2020). *Dive into deep learning*. URL http://d2l.ai.

[9] G. Landrum. *RDKit: Open-source cheminformatics*. URL http://www.rdkit.org.

[10] D. Mendez et al. (2019). *ChEMBL: towards direct deposition of bioassay data*. Nucleic Acids Res. 47, D930.

[11] A. Vaswani et al. (2017). *Attention is all you need*. Advances in neural information processing systems (pp. 5998–6008).

[12] N. Srivastava et al. (2014). *Dropout: a simple way to prevent neural networks from overfitting*. The Journal of Machine Learning Research, 15(1), 1929–1958.

[13] E. Jang, S. Gu, and B. Poole. (2016). *Categorical reparameterization with gumbel-softmax*. arXiv preprint arXiv:1611.01144.

[14] A.F.T. Martins and R.F. Astudillo (2016). *From softmax to sparsemax: A sparse model of attention and multi-label classification*. arXiv preprint arXiv:1602.02068.

[15] H. Inan, K. Khosravi, and R. Socher. (2016). *Tying word vectors and word classifiers: A loss framework for language modeling*. arXiv preprint arXiv:1611.01462.

[16] H. Jaeger. (2002) *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach*. Vol. 5. GMD-Forschungszentrum Informationstechnik Bonn.

[17] S. Hochreiter and J. Schmidhuber. (1997). *Long short-term memory*. Neural computation, 9(8), 1735–1780.

[18] K. Cho et al. (2014). *On the properties of neural machine translation: encoder-decoder approaches*. arXiv preprint arXiv:1409.1259.

[19] Y. Gal and Z. Ghahramani. (2016). *A theoretically grounded application of dropout in recurrent neural networks*. In NIPS, pp. 1019–1027.

[20] A. Holtzman et al. (2019). *The curious case of neural text degeneration*. arXiv preprint arXiv:1904.09751.

[21] M. Benhenda. (2017). *Chemgan challenge for drug discovery: can AI reproduce natural chemical diversity?* arXiv preprint arXiv:1708.08227.

[22] D. Rogers and M. Hahn. (2010). *Extended-connectivity* fingerprints. Journal of chemical information and modeling 50(5):742–754.

[23] D. Kingma and J. Ba. (2014). *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980.

[24] X. Glorot and Y. Bengio. (2010). *Understanding the difficulty of training deep feedforward neural networks*. In Proceedings of the thirteenth international conference on artificial intelligence and statistics, pages 249–256.

[25] G.R. Bickerton et al. (2012). *Quantifying the chemical beauty of drugs*. Nature Chemistry, 4, 90-98.

[26] Z. Dai et al. (2019). *Transformer-xl: Attentive language models beyond a fixed-length context*. arXiv preprint arXiv:1901.02860.

[27] N. Kitaev, Ł. Kaiser, and A. Levskaya. (2020). *Reformer: The efficient transformer*. arXiv preprint arXiv:2001.04451.

[28] K. He et al. (2015). *Deep residual learning for image recognition*. arXiv preprint arXiv:1512.03385.

[29] R. K. Srivastava, K. Greff, and J. Schmidhuber. *Highway networks*. arXiv:1505.00387, 2015.

[30] J. L. Ba, J. R. Kiros, and G. E. Hinton. *Layer normalization*. arXiv preprint arXiv:1607.06450, 2016.