Exploring the GDB-13 chemical space using deep generative models

Josep Arús-Pous†⊥*, Thomas Blaschke†∥, Silas Ulander§, Jean-Louis Reymond⊥, Hongming Chen†, Ola Engkvist†

Author Affiliations

† Hit Discovery, Discovery Sciences, IMED Biotech Unit, AstraZeneca, Gothenburg, Pepparedsleden 1, 43183 Mölndal, Sweden.

§ Medicinal Chemistry, Cardiovascular, Renal and Metabolism, IMED Biotech Unit, AstraZeneca, Gothenburg, Pepparedsleden 1, 43183 Mölndal, Sweden.

⊥ Department of Chemistry and Biochemistry, University of Bern, Freieistrasse 3, 3012 Bern, Switzerland.

∥ Department of Life Science Informatics, B-IT, LIMES Program Unit Chemical Biology and Medicinal Chemistry, Rheinische Friedrich-Wilhelms-Universität, Endenicher Allee 19C, D-53115 Bonn, Germany.

Corresponding Author
E-mail: josep.arus@dcb.unibe.ch; Address: Department of Chemistry and Biochemistry, University of Bern, Freieistrasse 3, 3012 Bern, Switzerland.
Keywords: Deep Learning, Chemical Space Exploration, Deep Generative Models, Recurrent Neural Networks, Chemical Databases.

Abstract: Recent applications of Recurrent Neural Networks enable training models that sample the chemical space. In this study we train RNN with molecular string representations (SMILES) with a subset of the enumerated database GDB-13 (975 million molecules). We show that a model trained with 1 million structures (0.1 % of the database) reproduces 68.9 % of the entire database after training, when sampling 2 billion molecules. We also developed a method to assess the quality of the training process using log-likelihood plots. Furthermore, we use a mathematical model based on the “coupon collector problem” that compares the trained model to an upper bound, which shows that complex molecules with many rings and heteroatoms are more difficult to sample. We also suggest that the metrics obtained from this analysis can be used as a tool to benchmark any molecular generative model.
Introduction

Finding novel molecules with specific properties is one of the main problems that drug discovery faces. One of the most common approaches to this is to explore chemical space by enumerating large virtual libraries, hoping to find a novel region of space containing useful structures. However, the drug-like chemical space is intractably large and a rough estimate would be at least $10^{23}$ molecules [1]. There are two classical approaches to exploring chemical space. One is to use implicit models, which do not store all molecules in a region of the chemical space but instead represent molecules indirectly. Techniques such as chemical space navigation by mutations [2] or creating reaction graphs have proven to be successful [3, 4]. The other more common way is to use explicit models. By searching public databases that contain molecules obtained from various sources, e.g. ChEMBL [5], new molecules of interest can be discovered. An alternative approach is the GDB project, a set of databases that exhaustively enumerate a part of the chemical space. For example, GDB-13 [6] and GDB-17 [7] are large databases that hold all drug-like molecules up to 13 and 17 heavy atoms ($\sim10^9$ and $\sim10^{11}$ molecules) respectively. Additionally, GDB-4c [8] is a database that enumerates all possible ring systems up to 4 rings. These databases include a wealth of novel structures of potential interest for drug discovery [9].

In recent years deep learning has been a major addition in machine learning. Problems that were difficult to tackle before are now successfully approached using deep learning, such as image classification [10], face recognition [11] or playing Go [12]. Recently there has been another step forward in the field with deep generative models, which generate content similar to that upon which they have been trained. Deep generative models have been successfully applied to music composition [13], image generation [14] and language translation [15]. These new methods are also being applied to chemical space exploration in a novel way [16]. When trained with a small
subset of molecules, these models generate molecules similar to the training set. Different types of neural networks such as Variational Auto-Encoders (VAE) [17], Recurrent Neural Networks (RNN) [18, 19] and Generative Adversarial Networks (GAN) [20] trained with string representations (SMILES) [21] from ChEMBL have proven to be successful at generating novel chemical space.

Despite the results obtained by previous research, the question as to how much of the chemical space surrounding the molecules in the training set can be generated by a RNN trained with SMILES remains unanswered. The Fréchet ChemblNet distance, [22] which compares a given generated chemical library with real molecule data from ChEMBL was recently proposed as a benchmark. However, the ChEMBL database has two main issues. First, it has a strong bias, as it only contains molecules obtained from literature. Second, it is very sparse, spanning a large chemical space with roughly 1.5 million molecules. The SMILES notation has also a known ambiguity: each molecule can have multiple different SMILES that uniquely represent it.

Here we aim to gain insight on how a RNN explores the chemical space and how the SMILES format affect it by training RNN with canonical SMILES sampled from the GDB databases. We use GDB-13, because this database has a large yet still manageable size (975 million molecules). Figure 1 illustrates the whole domain of possible outcomes from a RNN trained with SMILES. This domain changes during the training process: before training the RNN generates random strings, a few of which are going to be valid SMILES. After training, the generated strings are mostly valid SMILES that, to a large extent, belong to GDB-13. By computing how much of the whole GDB-13 a model can generate from a small subset and which molecules outside of the domain of GDB-13 are generated, the learning limitations are assessed. To do this, the results obtained from the trained model are compared to those from an abstract ideal model which
generates all GDB-13 molecules with uniform distribution. Any model, regardless of its architecture or input format, trained with a subset of GDB-13 can be compared to this ideal model in the same manner, thus creating a new way to benchmark the limitations of models prior to using them to explore chemical space.

Deep learning based molecular generation methods can be applied either to optimize an already existing chemical series or to find through scaffold hopping a completely novel chemical series. While for optimizing a chemical series, it is only necessary to investigate the local chemical space around the series, for scaffold hopping it is important to span the whole desirable chemical space and in addition not wasting time generating molecules outside the desirable chemical space. Therefore, the proposed benchmark will be especially important for scaffold hopping to ensure that the model explores as much as possible of the desired chemical space, while minimizing sampling of undesirable compounds.

**Figure 1:** Representation as an Euler diagram of the domain of a RNN trained with SMILES strings. The sets are the following, ordered by their size: All possible strings generated by an RNN
(red), all possible valid SMILES (yellow), all possible SMILES of GDB-13 molecules (light blue), all canonical SMILES of GDB-13 molecules (dark blue) and the training set (black). Note that the relative sizes of the different subsets do not reflect their true size.

**Methods**

**Recurrent neural networks**

A (feed-forward) neural network [23] (NN) is a machine learning architecture that maps a given input to some output result. After training with a set of predefined input-output examples (called the training set), the system modulates the outputs depending on the inputs given, having a similar behavior to the training set. The internal structure of the system is formed by a series of fully interconnected layers (formed by nodes), starting with the input layer, the hidden layers and ending with the output layer. This topology vaguely resembles a biological neural network, thus its name.

Recurrent Neural Networks [23] (RNNs) add additional complexity to the feed-forward ones, by converting the topology to a directed graph (which can have cycles). This allows the network to perform recursion and exhibit dynamic temporal behavior. This dynamic behavior creates persistence in the network, not dissimilar to memory. Importantly, a difference between RNNs and NNs is that, instead of having fixed-length input and output vectors, can be run sequentially. This allows networks to operate on sequences of inputs and thus enabling efficient parsing of content of varying length (one-to-many, many-to-one or many-to-many inputs-outputs).

The most common architecture used in RNNs is to connect layers with time-dynamic behavior to layers that normalize the input and the output to achieve an iterative behavior. For each iteration, the model receives two inputs: a vector of numbers and also a hidden state matrix (which contains information from the previous steps), and returns two outputs: an output vector and an updated hidden state matrix. For the next iteration the output and the hidden state from the previous
iteration is input. This is repeated until all the input sequences are added, or when the end conditions are met (i.e. outputting specific data).

Since the development of RNNs [24], the system was often unable to learn correctly when many recurrent layers were connected together or the input sequence was too long, due to problems such as vanishing and exploding gradients [25]. These were mitigated by using a very specific layer called a Long Short-Term Memory [26] (LSTM). Further optimization led to the Gated Recurrent Unit [27] (GRU), which has been demonstrated to produce similar results at a lower computational cost.

**Training a model with SMILES**

SMILES were discretized into tokens before inputting them to the RNN. Each atom was extracted as a token, taking special care with the multi-letter atoms “Br” or “Cl”. Moreover, all atoms between brackets, such as “[N+]” and “[O-]” were converted into only one token. The set with all the possible tokens is called the vocabulary.

After gathering the vocabulary, two special symbols were added: “^” and “$”, which represent the beginning and end of a sequence respectively. SMILES strings were then encoded using a series of one-hot vectors, each with as many binary positions as tokens in the vocabulary. The represented token having a “1” and the rest “0”. All the SMILES strings were encoded as a matrix with a “^” and “$” token added in the first and last position respectively.

The RNN architecture (Figure 2) used in this publication is similar to previous approaches [18, 19]. First an embedding layer [28] with 256 dimensions converts the discrete one-hot-encoded SMILES to a continuous representation. Then 3 layers composed of 512 GRU units comprise the bulk of the network. Lastly, a fully-connected linear layer performs softmax operation on the
output, making the values sum up to one so they can be used as a probability distribution with the same size as the vocabulary.

Figure 2: Example of a forward-pass of nicotine (CN1CCCC1c1ccnc1). The symbol sampled from the probability distribution at the stage $i$ is input at the stage $i + 1$. This, with the hidden state ($h_i$), enables the model to have time-dynamic behavior. Note that the probability distributions are not from real trained models and that the vocabulary used throughout this publication is much bigger.

For each RNN, two sets were collected beforehand. The training set is a 1 million molecule random sample of GDB-13 used to train the model. The validation set is another sample of 100,000 molecules not from the training set, used to evaluate the performance of the model during training.
The sampling process of the model is illustrated in Figure 2. First the “^” token is passed in and the RNN outputs a probability distribution for all the possible tokens. For the next token to be sampled, the RNN requires the previous token and hidden state (memory) to be inputted again. The process continues until a “$” symbol is outputted. Defining 

$$P(X_i = T_i | X_{i-1} = T_{i-1}, ..., X_1 = T_1)$$

as the probability of sampling token $T_i$ on step $X_i$ after having sampled tokens $T_{i-1} ... T_1$ on steps $X_{i-1} ... X_1$, the resulting probability on step $i$ is:

$$P(X_i = T_i, ..., X_1 = T_1) = P(X_1 = T_1) \cdot \prod_{k=2}^{i} P(X_k = T_k | X_{k-1} = T_{k-1}, ..., X_1 = T_1)$$

As the value would rapidly diminish to 0, due to hardware precision problems, logarithm (in our case natural logarithm) sums are used:

$$LL_i = -\ln P(X_i = T_i, ..., X_1 = T_1)$$

$$= -\ln P(X_1 = T_1) - \sum_{k=2}^{i} \ln P(X_k = T_k | X_{k-1} = T_{k-1}, ..., X_1 = T_1)$$

(Equation 1)

This value is called a log-likelihood (LL) and it gives a measure on how likely a sequence is to appear when randomly sampling the model. Its range is $[0, +\infty)$ with higher values corresponding to lower probabilities.

As in previous research [18, 19], backpropagation with the ADAM optimizer was used to train the RNN. The goal is to minimize a cost function $J(w)$ for all molecules in the training set. To achieve that, it calculates from the last to the first step the average of the $J(w)$ of a set of sequences (a batch). From this, a gradient is calculated which can be used to iteratively fit the model to the training data. Formally, the loss function is the partial LL up to position $i$:

$$J(w) = LL_i$$
The teacher’s forcing [29] method was used. In this method the likelihood calculation on step $i$ is calculated from the previous tokens in the training SMILES and not from the possibly wrong token of the untrained RNN. This allows the RNN to learn the information in the training set faster and more reliably.

The training data was passed to the RNN multiple times: each iteration, called an epoch, all compounds in the set were input to the RNN. To enhance the learning process, the learning rate (LR) was reduced after every epoch. This hyperparameter controls the optimization speed of the learning process, higher LR imply faster learning but less refined solutions. After some early testing it was observed that a LR greater than $10^{-3}$ and smaller than $10^{-5}$ have no effect on the training whatsoever, so the LR changes from $10^{-3}$ to $10^{-5}$, being multiplied by a constant every epoch.

**Ideal model**

RNN-based models must learn how to generate SMILES and how to create molecules that appear in GDB-13. An ideal model is an abstract model that samples molecules from GDB-13 and only from GDB-13. Formally, the probability of sampling any molecule in the ideal model follows a uniform probability distribution with $p = \frac{1}{|GDB-13|} = 1.02 \cdot 10^{-9}$. Due to the probabilistic nature of RNNs, no trained model will be able to have the same behavior, thus it serves as an upper bound.

We can calculate the expected number of times GDB-13 needs to be sampled to obtain 100% of the database. This problem is commonly known in mathematics as the “coupon collector problem” [30]. It was originally used to calculate the number of coupons (or stickers) that are needed to be bought to be able to obtain the full collection, knowing that every time a coupon is bought it is sampled with replacement from a distribution containing all possible coupons. Formally, for a uniform distribution with $n$ coupons:
\[ E[T_u] = n \cdot H_n \approx n(\ln(n) + \gamma) + \frac{1}{2} \]

(Equation 2)

Where \( H_n \) is the n-th harmonic number and \( \gamma \) is the Euler-Mascheroni constant. By fitting this to the GDB-13 we would need to sample on average 20,761,554,747 SMILES. In the case of a non-uniform distribution, the previous number is a lower bound of \( E[T] \), and the upper bound is the following:

If we set \( p_k < p_{k+1} \forall k \in [1, n] \) we define:

\[ N = \sum_{k=1}^{n} \frac{p_i}{p_1} \]

Notice that \( N \geq n \) and that each \( p_i \) is subdivided into many smaller buckets, all of them of the smallest \( p_i \). This allows the conversion from a non-uniform to a uniform distribution. Hence, the upper bound for the expected value is:

\[ E[T_{nu}] < N \cdot H_N \approx N(\ln(N) + \gamma) + \frac{1}{2} \]

(Equation 3)

The expected value can tend to infinity for distributions where \( \exists p_k \to 0 \). This is generally the case for all the trained models in this publication. We can also obtain the expected fraction of a database with \( n \) coupons if \( k \) were sampled from the ideal model.

Given a probability distribution where \( \sum_{i=1}^{n} p_i = 1 \), Let \( X \) be the number of different coupons collected when sampling \( k \) times a set with \( n \) coupons. The probability of sampling \( k \) times without picking any specific coupon is \( P(t_i) = (1 - p_i)^k \). With \( P(\bar{t}_i) = 1 - X_i \) as the complementary probability. The expected number of different items when sampling \( k \) times is:
$$E[X] = \sum_{i=1}^{n} P(\xi_i) = n - \sum_{i=1}^{n} (1 - p_i)^k$$

The fraction of the database can then be obtained as:

$$fraction = \frac{n - \sum_{i=1}^{n} (1 - p_i)^k}{n}$$

(Equation 4)

Notice that also when $\exists \ p_k \to 0$ the whole database can never be fully sampled regardless of the sample size. For a uniform distribution, the previous equation can be simplified to:

$$fraction_{\text{uniform}} = 1 - (1 - p)^k$$

In the case of a sample of 2 billion molecules from the ideal model the average fraction of molecules sampled would be 0.8712.

**Sampling SMILES from a model**

To be able to evaluate how much of GDB-13 can be reliably sampled from a model, it must be sampled at least 21 billion times. This has an unfeasible computational cost. For this reason, samples of 2 billion molecules were performed, which account for approximately 10% of the optimal sample size. After each sample, several tests were done: the database was checked for duplicates, for invalid SMILES, for non-canonical SMILES and was intersected with GDB-13, yielding 2 subsets: IN and OUT of GDB-13.

**PCA plots with MQN**

PCA plots were based on the method described previously in literature [31]. The 42-dimension MQN fingerprint[32] was calculated with the JChem Library 18.22.0 from ChemAxon (www.chemaxon.com) for each of the molecules in the dataset. Then, without any normalization or standardization, a Principal Component Analysis (PCA) was performed on the 42-dimensional resulting dataset. The two first principal components were selected and normalized to values
between 0 and \( w \) or \( h \) and molecules were organized in buckets. Each bucket represents a pixel \((x, y)\) in the resulting \( w \times h \) plot with a black background. A descriptor was also calculated for all molecules in each bucket and the average and count were calculated and normalized to the range [0,1]. To color the pixels, the Hue-Saturation-Value (HSV) format was used with the normalized average descriptor as hue, a fixed value of 1.0 as the saturation and \( value = \min(0.25, \log_{10}(\text{count}_{\text{norm}})) \). With this setup, the pixels that have low count are gradually merged with the background and those that have the highest counts stand out.

**Labelling sampled molecules out of GDB-13**

Sampled molecules not included in GDB-13 were labeled with the topological and chemical filters used in the enumeration process of GDB-13 that they broke [6, 33]. The molecules with disallowed topology were labelled the following way: carbon skeletons from all the molecules in GDB-13 were calculated and compared to the carbon skeletons for each sampled molecule, labelling the molecules whose skeleton was not in GDB-13. All tautomers for all the molecules were calculated with MolVS [34] 0.1.1. For each molecule, if one tautomer was part of GDB-13, the molecule was labelled as a tautomer. Molecules with disallowed functional groups, heteroatom configurations or bonds were detected using SMARTS.

**Technical details**

All the programming, except noted, was done in Python 3.6 using RDKit [35] version 2018.03 as the chemistry toolkit and PyTorch [36] 0.4.1 as the deep learning library. Stochastic gradient descent was used for training with the ADAM[37] optimizer with parameters \( \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8} \) and a batch size of 128.

The GDB-13 database was obtained from the [gdb.unibe.ch](http://gdb.unibe.ch) website and preprocessed with RDKit to obtain canonicalized SMILES and to filter molecules impossible to read with the toolkit.
The final size of the database was 975,820,187 molecules. Data processing and PCA calculation were done with Apache Spark [38] 2.3.1 and all datasets were stored in Apache Parquet files. All plots, including the PCA maps, were created with Matplotlib [39] and Seaborn [40]. The Jensen-Shannon Divergence was calculated with an in-house script using SciPy [41].

All calculations were performed in CentOS 7.4 with Tesla V-100 (Volta) graphics cards and CUDA 9.

| Operation                        | CPUs | RAM   | GPU | Time          |
|----------------------------------|------|-------|-----|---------------|
| Training a model                 | 4    | 32 GB | 1   | 8 min. / epoch|
| Sampling molecules               | 4    | 32 GB | 1   | 33 million / h.|
| Annotating 2B molecules          | 32   | 256 GB| 0   | 24 h.         |

Table 1: Computational resources and cost associated with training and sampling the model and annotating a 2B sample.
Results and discussion

Using log-likelihood plots to guide the training process

A model was trained with a training set of 1 million compounds randomly sampled from GDB-13. A first way to assess the quality of the sampled molecules is to check the percentage of valid molecules (Figure 3c). This metric has often been used to train models [18, 19], but in the case of the GDB databases it proves to be insufficient, as it is always over 96.5%. To view the progress of the training process, log-likelihoods (LLs) of the SMILES in the training, validation and sampled sets were calculated after training the model each epoch. These LLs were plotted together as histograms every 25 epochs (Figure 3a). Also, the Jensen-Shannon Divergence (JSD) between the three plots has been calculated (Figure 3b). This measure allows the quantification of the differences between each pair of distributions.

Figure 3 plots are interpreted as follows: after epoch 1, the sampled set LL distribution has the lowest average (higher probability) and the other two sets are extremely similar and have a higher LL (lower probability). This means that the model isn’t completely trained, as the SMILES strings sampled are only a subset of the ones in the training set. Between epochs 25 to 50, the distributions become more similar, and around epochs 50-100 the three plots match as much as possible, as can be seen both in (a) and in (b). When all the plots are similar it is equally probable to sample a SMILES from the training set as it is a SMILES outside it, implying that a higher percent of the database can be sampled. After this, the training set LL distribution becomes more similar to the sampled set while the validation set has higher LL. This indicates overtraining, as the model will sample a molecule from the training set with a higher probability than a molecule from the validation set. This trend becomes more pronounced in later epochs.
**Figure 3:** Metrics used to evaluate the training process. The red line at epoch 70 represents the chosen epoch used in further tests. The log-likelihood (LL) is calculated with natural logarithms.

**a)** 10 LL plots of the training, validation and sampled sets every 25 epochs (from 1 to 200) and the chosen epoch (70).

**b)** JSD plot between the three LL distributions from the previous section for each of the 200 epochs.

**c)** Percent of valid molecules in each epoch. Notice that the plot already starts at around 96.5%.

**d-e)** Mean (d) and variance (e) of the three distributions from section (a).
Note that spikes around epochs 1-20 are statistic fluctuations common in the beginning of the training process of a RNN, when the learning rate is high.

To discern whether the model is uniform, the mean ($\mu$) and the variance ($\sigma^2$) of the LL distributions have been calculated after each training epoch (Figure 3c-d). Knowing that the uniform model LL plot has $\sigma^2 = 0$ and $\mu = -\ln \left( \frac{1}{|GDB-13|} \right) = 20.7$, the variance and the mean of the validation set should be as similar to these values as possible. This happens around epochs 60-90. Choosing the model trained up to any of this range yields a model with mostly indistinguishable behavior.

To further understand this method, 2 billion SMILES strings were sampled every 5 epochs (totaling 80 billion). As can be seen in Figure 4a, the total percent of generated molecules including repeats that are part of GDB-13 always increases, but in Figure 4b the percent of unique molecules generated that are in GDB-13 is maximal at around epoch 70 (68.9 %) and starts to decrease again after epoch 100. Also, the sampled molecules not included GDB-13 steadily decrease during the whole training. These results match the LL plots and the variance plot in Figure 3a and e. The best models are between epoch 70 and 100. Having a model representing a more uniform sampling (epoch 70) conflicts with having a more focused sampling (epoch 100). Depending on the specific needs for a given project a different epoch should be chosen, yet the differences are very small. Epoch 70 was chosen for future experiments with this model, because a more uniform model was desired.
Figure 4: Results from sampling 2 billion SMILES from the 1M model every 5 epochs (from 1 to 195). The red line at epoch 70 represents the chosen epoch for further tests. a) Percent of the total sample (2B) that are valid SMILES, canonical SMILES, in GDB-13 and out of GDB-13. Solid lines represent all SMILES sampled, including repeats, whereas dotted lines represent only the unique molecules obtained from the whole count. b) Close-up percentage of GDB-13 obtained every 5 epochs. Notice that the plot starts at around 54% and that the drop around epoch 10 correlates with the training fluctuations already mentioned in Figure 3.

For any molecule there are many SMILES that uniquely represent it. In Figure 1 the light and dark blue sets represent the number of possible SMILES for all the molecules and only one canonical SMILES for each molecule respectively. In the ideal model, only the canonical SMILES for each molecule are generated. Figure 4a shows (pink) that 85.6 % of the SMILES in epoch 70 were generated directly as canonical, implying that the model is able to learn the canonical form of most of the generated SMILES. Notice also that the number of unique canonical SMILES
decreases steadily. This is correlated with the model not being uniform, and this trend is further pronounced after epoch 100, as the molecules from the training set are generated more often.

**Understanding the diversity of the generated molecules**

25 models with the same parameters as in the previous section were trained with a different 1M random sample obtained from GDB-13. The probability of sampling each molecule from GDB-13 is averaged and molecules not generated by any model have a higher chance to be problematic due to the limitations of the model and not by chance.

For each model, 2 billion molecules were sampled in epoch 70 (summing up to 50 billion molecules), repeated molecules were filtered and the whole sample was separated between molecules contained and not contained in GDB-13. Note that the number of molecules needed to sample from the ideal model to obtain 100% of the database on average is around 21 billion (Equation 2), much less than the 50 billion molecules sampled in this experiment. The frequency for each molecule in GDB-13 was computed, which is the number of times from 0 (not sampled in any model) to 25 (sampled in all models) each molecule was uniquely sampled from each model.

In the ideal model, each sample can be considered as a Bernoulli trial with $p = 0.8712$ (see Equation 4), so the distribution of the frequency would follow a binomial distribution with $n = 25$, $k = 2 \cdot 10^9$ and $p = 0.8712$. Figure 5a shows that the two distributions have a different mean (17.1 and 21.8) and mode (20 and 22) and the distribution obtained from the RNN models has an extremely long tail. Moreover, 5,720,928 (0.6 %) molecules have never been sampled by any model. Notice also in Figure 5b that frequency is heavily correlated with the average likelihood for each molecule obtained from every model.
Figure 5: a) Histograms of the frequency of the RNN models (orange) and the theoretical (binomial) frequency distribution of the ideal model (blue). b) Histograms of the average LL per molecule (from the 25 models) for molecules with frequency 0, 5, 10, 15, 20 and 25 computed from a sample of 5 million molecules from GDB-13.

Analysis of the sampled molecules included in GDB-13

PCA plots of the MQN fingerprint were performed with a sample of GDB-13 stratified by frequency (Figure 6). Figure 6a, shows that there is a difference between the molecules that have lower (top-right) and higher (bottom-left) frequency. Nevertheless, the density plot (Figure 6b) shows that the most densely packed regions are at the center and occupied by molecules with both a high and a low frequency. Additional PCA plots were generated with some key descriptors that help pinpointing the different regions of the chemical space. Figure 6c shows that pixels at the right have mostly cyclic bonds, implying more rings, less sidechains and linkers. This area is mostly covered by molecules that have low frequency. Moreover, Figure 6d shows that pixels at the top have more heteroatoms. This closely matches the top lighter area in Figure 6a, which features molecules with low frequency.
Figure 6: a-f) MQN PCA plots (Explained variance: $PCA_1 = 51.3 \%, PCA_2 = 12.2 \%$) calculated from a 130 million stratified sample of GDB-13 with 5 million molecules from each frequency value (0 - 25) colored by different descriptors. In all plots each pixel represents a group of similar molecules and its color represents the average value of a given descriptor. The colors rank from minimum to maximum: dark blue, cyan, green, yellow, orange, red and magenta. Each plot has the numeric range (min – max) between brackets after its title. Plots are colored by: a) Number of trained models that generate each molecule. b) Occupancy of every pixel. c) Number of cyclic bonds. d) Number of carbon atoms.
From the previous plots, molecules with many heteroatoms or complex topologies have a lower probability of being sampled than molecules with less rings and more carbon atoms. However, Figure 6b also shows that most of these structures are in lower density zones of the database, which implies that are only a small part of the database. In Table 2, 24 fragment-like molecules with frequency 0, 5, 10, 15, 20 and 25 were selected from GDB-13 and shows that molecules with lower frequency have a tendency of having a more complex structure, especially more cyclic bonds, although it is not possible to separate them clearly.

To further understand how molecules are generated, the composition of the SMILES was analyzed. As shown in Figure 7a (dashed orange line), the 1-gram (token) count distribution is exponential and mostly features C (40 %). In order, less featured tokens are 1, N, =, (, ), O and 2. The rest of the tokens sum up to less than 7% of the total. SMILES representing simple topologies use mostly the tokens enumerated before and molecules that have complex shapes tend to have more rings, so they have less common tokens, such as 3,4, …, 7. The frequency of the molecules containing each token is also plotted in Figure 7a, showing that it correlates with the counts. Note especially, marked in red in Figure 7a, the numeric tokens starting from 4 tend to have a significantly lower average frequency than the neighboring tokens. This means that molecules in GDB-13 with 4 rings or more are significantly less probable to be sampled than others. One explanation is that these tokens appear in pairs in any valid SMILES, which indicates that learning how to create a correct molecule SMILES with these tokens is much more difficult than with other equally frequent tokens, as both tokens in each pair must be correctly positioned to each other. Additionally, molecules in GDB-13 (max. 13 heavy atoms) with more than 3 rings have extremely complex topologies. When performing the same analysis for 2-grams same interpretation applies (Figure 7b): the count is correlated with the average frequency and the most frequent 2-grams (CC,
C1, C(, )C, C=) match the SMILES of simple molecules and the less frequent (5o, 3[N+], 7O, 7(, 72) match exclusively molecules with complex topologies and several rings. This implies that the n-grams that appear less times in the database, i.e. in the training set, are not learned correctly and thus have a lower probability of being sampled. Therefore, molecules that contain an increasing number of low probability n-grams in their canonical SMILES, will progressively have a lower probability of being sampled (Equation 1).

**Figure 7:** Plots of the frequency (left y axis) and the percent in database (right y axis) of 1 and 2-grams in the canonical smiles of all GDB-13 molecules. The plot is sorted by the percent in database. **a)** Plot with the 1-grams (tokens). In blue the mean frequency and in orange the percent of 1-grams in database. Notice that the numeric tokens have been highlighted in red. **b)** Plot with the 2-gram mean frequency (blue) and percent (dashed orange). As the number of 2-grams is too large (287), the x axis has been intentionally left blank and the mean frequency has been smoothed by an average window function size 8.
Table 2: A selection of 24 fragment-like molecules obtained from GDB-13 with frequency 0, 5, 10, 15, 20 and 25. The molecules are sorted top to bottom by frequency and left to right by average log-likelihood (LL) of the 25 models. A random sample of 10 million molecules annotated with the frequency and the average LL is available for download (http://gdb.unibe.ch/downloads).

Analysis of the sampled chemical space outside of GDB-13

All the SMILES outside of GDB-13 generated by the 25 models were joined obtaining a database with 10,084,412,477 molecules. After filtering repeated molecules, a set with 2,979,671,366 unique molecules was obtained, from which a sample of 3 million was used for further research. Each molecule was then labelled with the constraints used to enumerate GDB-13 [6, 33] that it breaks (see methods). Figure 8 includes a plot with the percent of molecules that break each constraint (Figure 8a) and another histogram with the number of constraints broken per molecule (Figure 8b). The most common broken constraint, not allowed functional groups (26.2 %), is the most complex one to learn, as any given functional group can have multiple SMILES strings, depending on where it is positioned in the molecule, thus making more difficult to learn the string patterns to avoid. Also, 19.8 % of the molecules have a graph that was filtered during the GDB-13 enumeration process, which correlates with the problems encountered when generating molecules with complex graph topologies: the model is not able to correctly learn the underlying graph topologies of the molecules. Additionally, due to the probabilistic nature of the model, 17.5 % of the molecules generated outside of GDB-13 have more than 13 heavy atoms. Heteroatom / Carbon ratios used to create GDB-13 are generally followed (10.9 %) and there are a similar number (10.1 %) of molecules with disallowed neighboring heteroatom configurations. These constrains can easily be learnt by the model, as they have very little topological complexity compared to the previous two. For the same reason, 9.4 % of the database are tautomers of molecules existing in
GDB-13 and less than 7 % of the molecules have problems with double or triple bonds. Interestingly, the miscellaneous category (22.7 %) includes all molecules that are not in GDB-13 and that have broken none of the previous filters. This occurs partially due to problems with the chemical library used (GDB-13 was created with JChem from 2008 and this research uses RDKit from 2018) and because GDB-13 is not completely exhaustive: many molecules would never have been obtained with the enumerative approach used to create the database. Lastly, Figure 8b shows that 72 % of the molecules only break one constraint, hence the chemical space generated outside of GDB-13 is very near the space represented by GDB-13.

**Figure 8**: Distribution of a sample of 3 million molecules obtained from all the outside of GDB-13 sampled by the RNN model. **a)** Histogram of the GDB-13 constraints broken by each molecule. Notice that a molecule can break more than one constraint. **b)** Distribution of the number of GDB-13 constraints broken by each molecule.
Conclusions

This study shows that a large amount of chemical space can be sampled with generative models that are trained only with a very small sample of that chemical space. Specifically, we generate 68.9% of GDB-13 by only training with 0.1%. The model is not only capable of learning basic chemistry (e.g. valency, ring structures) but also to follow complex constraints applied during the GDB-13 enumeration process, such as heteroatom ratios and positioning of double and triple bonds. More difficult constraints, e.g. complex graph topologies or not allowed functional groups are more difficult to learn mostly due to the limitations of the SMILES notation. Besides, we developed a computationally cheap method to monitor and assess the quality of the training process using LL plots. Additionally, we performed sampling of the model and compared the results with those from the ideal model. We obtained that although most of the problematic molecules have a tendency of having more cyclic bonds and heteroatoms, the main difference arises from the SMILES. Moreover, these metrics can be used as a benchmarking tool for molecular generative models. The ideal model, which uniformly samples molecules from and only from GDB-13, sets a limit (87.5%) to the amount of GDB-13 mapped with a 2 billion sample. We encourage researchers to try this method on models with different architectures or input formats and compare the results. This may lead to a better understanding of molecular generative models.

Abbreviations

ADAM - ADApative Moment estimation
GAN – Generative Adversarial Network
GDB – Generated DataBase
GRU – Gated Recurrent Unit
HSV – Hue-Saturation-Value
LL – Log-Likelihood
LR – Learning Rate
LSTM – Long Short-Term Memory
MQN – Molecular Quantum Numbers
NN – Neural Network
PCA – Principal Component Analysis
RNN – Recurrent Neural Network
SMARTS - SMiles ARbitary Target Specification
SMILES – Simplified Molecular-Input Line-Entry System
VAE – Variational Auto-Encoder

Declarations

Availability of Data and Materials

One sample model trained with GDB-13 and the software used to train and sample it described in this publication is available through a Github repository (https://github.com/undeadpixel/reinvent-gdb13). The GDB-13 database and a 1 million random sample annotated with frequency and average log-likelihood is available through the Reymond group website (http://gdb.unibe.ch/downloads).

Competing interests

The authors declare that they have no competing interests.

Funding

Josep Arús-Pous and Thomas Blaschke are supported financially by the European Union’s Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement no. 676434, “Big Data in Chemistry” (“BIGCHEM,” http://bigchem.eu).
**Author Contributions**

Josep Arús-Pous designed and performed the research, wrote the software and wrote the manuscript. Thomas Blaschke co-wrote the software, reviewed and edited the manuscript. Silas Ulander performed the mathematical analysis. Ola Engkvist, Hongming Chen and Jean-Louis Reymond supervised the project. All authors have read and approved the manuscript for publication.

**Acknowledgements**

The authors would like to acknowledge Thierry Kogej and Christian Tyran for their scientific insight and Laurianne David, Oliver Laufkötter, Noé Sturm and Michael Withnall for their help in reviewing the publication.
References

1. Ertl P (2003) Cheminformatics analysis of organic substituents: Identification of the most common substituents, calculation of substituent properties, and automatic identification of drug-like bioisosteric groups. J Chem Inf Comput Sci 43:374–380

2. Van Deursen R, Reymond JL (2007) Chemical space travel. ChemMedChem 2:636–640

3. Hartenfeller M, Zettl H, Walter M, Rupp M, Reisen F, Proschak E, Weggen S, Stark H, Schneider G (2012) Dogs: Reaction-driven de novo design of bioactive compounds. PLoS Comput Biol 8:e1002380

4. Andersen JL, Flamm C, Merkle D, Stadler PF (2014) Generic strategies for chemical space exploration. Int J Comput Biol Drug Des 7:225

5. Gaulton A, Bellis LJ, Bento AP, et al (2012) ChEMBL: A large-scale bioactivity database for drug discovery. Nucleic Acids Res 40:D1100–D1107

6. Blum LC, Reymond JL (2009) 970 Million druglike small molecules for virtual screening in the chemical universe database GDB-13. J Am Chem Soc 131:8732–8733

7. Ruddigkeit L, Van Deursen R, Blum LC, Reymond JL (2012) Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17. J Chem Inf Model 52:2864–2875

8. Visini R, Arús-Pous J, Awale M, Reymond JL (2017) Virtual Exploration of the Ring Systems Chemical Universe. J Chem Inf Model 57:2707–2718

9. Reymond JL (2015) The Chemical Space Project. Acc Chem Res 48:722–730
10. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. Proc IEEE Comput Soc Conf Comput Vis Pattern Recognit 07–12–June:1–9

11. Taigman Y, Yang M, Ranzato M, Wolf L (2014) DeepFace: Closing the gap to human-level performance in face verification. Proc IEEE Comput Soc Conf Comput Vis Pattern Recognit 1701–1708

12. Silver D, Huang A, Maddison CJ, et al (2016) Mastering the game of Go with deep neural networks and tree search. Nature 529:484–489

13. Hadjeres G, Pachet F, Nielsen F (2016) DeepBach: a Steerable Model for Bach Chorales Generation.

14. Garg S, Rish I, Cecchi G, Lozano A (2017) Neurogenesis-inspired dictionary learning: Online model adaption in a changing world. IJCAI Int Jt Conf Artif Intell 1696–1702

15. Johnson M, Schuster M, Le Q V., et al (2016) Google’s Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. doi: 10.3115/v1/P15-1002

16. Chen H, Engkvist O, Wang Y, Olivecrona M, Blaschke T (2018) The rise of deep learning in drug discovery. Drug Discov Today 23:1241–1250

17. Blaschke T, Olivecrona M, Engkvist O, Bajorath J, Chen H (2018) Application of Generative Autoencoder in De Novo Molecular Design. Mol Inform. doi: 10.1002/minf.201700123

18. Olivecrona M, Blaschke T, Engkvist O, Chen H (2017) Molecular de-novo design through
deep reinforcement learning. J Cheminform. doi: 10.1186/s13321-017-0235-x

19. Segler MHS, Kogej T, Tyrchan C, Waller MP (2018) Generating focused molecule libraries for drug discovery with recurrent neural networks. ACS Cent Sci 4:120–131

20. Sanchez-Lengeling B, Outeiral C, Guimaraes GL, Aspuru-Guzik A (2017) Optimizing distributions over molecular space. An Objective-Reinforced Generative Adversarial Network for Inverse-design Chemistry (ORGANIC). ChemRxiv 1–18

21. Weininger D (1988) SMILES, a Chemical Language and Information System: l: Introduction to Methodology and Encoding Rules. J Chem Inf Comput Sci 28:31–36

22. Preuer K, Renz P, Unterthiner T, Hochreiter S, Klambauer G (2018) Fréchet ChemblNet Distance: A metric for generative models for molecules. doi: 10.1021/acs.jcim.8b00234

23. Schmidhuber J (2015) Deep Learning in neural networks: An overview. Neural Networks 61:85–117

24. Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities. Proc Natl Acad Sci 79:2554–2558

25. Hochreiter S, Bengio Y, Frasconi P, Schmidhuber J (2009) Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies. A F Guid to Dyn Recurr Networks. doi: 10.1109/9780470544037.ch14

26. Hochreiter S, Schmidhuber J (1997) Long Short-Term Memory. Neural Comput 9:1735–1780

27. Cho K, van Merrienboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y
(2014) Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. doi: 10.3115/v1/D14-1179

28. Rahman N, Borah B (2016) A survey on existing extractive techniques for query-based text summarization. 2015 Int Symp Adv Comput Commun ISACC 2015 98–102

29. Williams RJ, Zipser D (1989) A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. Neural Comput 1:270–280

30. Laplace P-S (1814) Chapitre II: De la probabilité des événements composés d’événements simples dont les possibilités respectives sont données (4). In: Théorie Anal. des Probab., 2nd ed. Mme. Ve. Courcier, Paris, pp 191–201

31. Blum LC, Van Deursen R, Reymond JL (2011) Visualisation and subsets of the chemical universe database GDB-13 for virtual screening. J Comput Aided Mol Des 25:637–647

32. Nguyen KT, Blum LC, van Deursen R, Reymond J-L (2009) Classification of Organic Molecules by Molecular Quantum Numbers. ChemMedChem 4:1803–1805

33. Fink T, Reymond JL (2007) Virtual exploration of the chemical universe up to 11 atoms of C, N, O, F: Assembly of 26.4 million structures (110.9 million stereoisomers) and analysis for new ring systems, stereochemistry, physicochemical properties, compound classes, and drug discove. J Chem Inf Model 47:342–353

34. Swain M, JoshuaMeyers (2018) mcs07/MolVS: MolVS v0.1.1. doi: 10.5281/ZENODO.1217118

35. Landrum G (2014) RDKit: Open-source cheminformatics. http://www.rdkit.org/
36. Paszke A, Chanan G, Lin Z, Gross S, Yang E, Antiga L, Devito Z (2017) Automatic differentiation in PyTorch. Adv Neural Inf Process Syst 30 1–4

37. Lee D, Myung K (2017) Read my lips, login to the virtual world. 2017 IEEE Int Conf Consum Electron ICCE 2017 434–435

38. Zaharia M, Franklin MJ, Ghodsi A, et al (2016) Apache Spark. Commun ACM 59:56–65

39. Hunter JD (2007) Matplotlib: A 2D graphics environment. Comput Sci Eng 9:99–104

40. Waskom M, Botvinnik O, O’Kane D, et al (2018) mwaskom/seaborn: v0.9.0 (July 2018). doi: 10.5281/ZENODO.1313201

41. Virtanen P, Gommers R, Burovski E, et al (2018) scipy/scipy: SciPy 1.1.0. doi: 10.5281/ZENODO.1241501