DeepSMILES: An adaptation of SMILES for use in machine-learning of chemical structures

Noel M. O'Boyle,[1] Andrew Dalke[2]

[1] NextMove Software, Unit 23 Cambridge Science Park, Cambridge CB4 0EY, UK
[2] Andrew Dalke Scientific AB, SE-461 30 Trollhättan, Sweden

Email: noel@nextmovesoftware.com, dalke@dalkescientific.com
Twitter: @baoilleach
Code: https://github.com/nextmovesoftware/deepsmiles

Abstract

Background

There has been increasing interest in the use of deep neural networks for *de novo* design of molecules with desired properties. A common approach is to train a generative model on SMILES strings and then use this to generate SMILES strings for molecules with a desired property. Unfortunately, these SMILES strings are often not syntactically valid due to elements of SMILES syntax that must occur in pairs.

Results

We describe a SMILES-like syntax called DeepSMILES that addresses two of the main reasons for invalid syntax when using a probabilistic model to generate SMILES strings. The DeepSMILES syntax avoids the problem of unbalanced parentheses by only using close parentheses, where the number of parentheses indicates the branch length. In addition, DeepSMILES avoids the problem of pairing ring closure symbols by using only a single symbol at the ring closing location, where the symbol indicates the ring size. We show that this syntax can be interconverted to/from SMILES with string processing without any loss of information, including stereo configuration.

Conclusion

We believe that DeepSMILES will be useful, not just for those using SMILES in deep neural networks, but also for other computational methods that use SMILES as the basis for generating molecular structures such as genetic algorithms.

Introduction

There has been a recent surge of interest in the application of Deep Neural Networks (DNNs) to the *de novo* design of molecules with desired properties. DNNs are used to create generative models based on training data, and these models are used to create new molecules. One approach, introduced by Gómez-Bombarelli et al [1], trains a Variational Auto Encoder (VAE) that maps SMILES strings (Weininger [2]) into a high-dimensional latent space; after this, points in that space, typically found by an optimization or search procedure, may be decoded back to SMILES (for example, Kusner et al [3]). An alternative approach is to build the model using a Recurrent Neural Network (RNN) with Long Short Term Memory (LSTM) (Segler et al [4], Ertl et al [5], Olivecrona et al [6]) and generate SMILES using this probabilistic model.

However, there is no guarantee that the resulting SMILES strings will be valid or even if they are, that they will represent a reasonable molecule structure. The percentage valid SMILES generated that represent reasonable molecules has been reported as ranging from 7% [3] to 80% [1]. These values can only be interpreted in the context of the diversity and relevance of the molecules generated (nor is it always clear what is meant by "valid molecule" or "valid SMILES"), but nonetheless illustrate the difficulties encountered.

These validity problems can be broken down into those related to the semantics of SMILES strings, versus those that involve invalid syntax. Typically, the former refer to atom valences that are unlikely to occur, or indeed never occur, in drug-like molecules; for example, a 5-coordinate carbon. Here we focus instead on the problem of invalid syntax, the majority of which involve either parenthesis notation for branches or ring closure notation.

In SMILES syntax, branches are indicated by balanced pairs of parentheses, an open parenthesis followed at some point by a corresponding close parenthesis. An example is the SMILES string "CCC(OC)CC)SC(ON" which is invalid although it contains an equal number of open and closed parentheses as the second close parenthesis does not have a preceding matching open parenthesis. Ring closures are indicated by a pair of identical ring closure symbols (which we will refer to as the ring-opening symbol and ring-closing symbol based on their order in the string), usually a single digit but which may be two digits preceded by a percent symbol; typically the first ring opening symbol in a SMILES string will use the digit 1, the next 2, and so on, although ring closure symbols may be reused within a SMILES string once a ring is closed. An example of an invalid SMILES due to unmatched ring closure symbols is C1CCCC2.

Some approaches to generating SMILES via machine-learning methods use a grammatical definition of the SMILES syntax to ensure that the SMILES generated are valid according to the grammar. Unfortunately, while the context-free grammar used (typically based on that by one of the authors, AD [7]) does ensure that brackets are matched, it does not (and cannot) enforce the requirement for ring closure symbols to be paired. Thus, for example, the approach of Naruki et al [8] using grammatical evolution or that of Kusner et al [3] using a grammar variational autoencoder still yield problematic SMILES. More recent work by Dai et al [9] uses a syntax-directed variational autoencoder that not only addresses this problem by incorporating a constraint on ring closure symbols, but goes further and incorporates a maximum valence constraint on the atoms. Also relevant here is the work by Janz et al [10] that uses a DNN as a validator during the decoding of SMILES from a latent space of a variational autoencoder.

By definition, the problems described only affect approaches that generate molecular structures as SMILES. To our knowledge the only study that has attempted to use an alternative line notation is that by Gómez-Bombarelli et al [1], where InChI strings [11] were tested but were found to perform "substantially worse" than SMILES. There are approaches that entirely avoid these problems by working directly on the molecular graph (Kearnes et al [12], Simonovsky and Komodakis [13], Li et al [14], You et al [15]) or on a reduced graph representation (Jin et al [16]). Nonetheless, the current popularity of SMILES-based generative models means that a solution to the SMILES syntax problems above would be beneficial.

The approach we propose is to transform a SMILES string to another syntax that conveys identical information but is more suited to machine-learning, a syntax we will refer to as DeepSMILES. Having converted the training set to DeepSMILES, these strings (and not the original SMILES) are then used as input to the machine-learning method. This process is then carried out in reverse to interpret the output of a generative model, by converting the generated DeepSMILES strings to SMILES strings for subsequent processing or display.

DeepSMILES addresses the syntax problems described above as follows:

1. DeepSMILES uses a single ring closure digit instead of two. Thus unmatched ring closure digits are not possible.
2. DeepSMILES avoids the use of paired parentheses by adopting a postfix notation, where one or more close parentheses are used to indicate branch length.

The two notations described above can be applied independently. As a result, any application of DeepSMILES should specify which transformations were applied.

A pure Python module is provided that interconverts between SMILES and DeepSMILES. Only minimal changes should be necessary to adapt existing workflows based around standard cheminformatics libraries.

Method

Transforming cycles

In regular SMILES, cycles are described using a pair of corresponding 'ring closure symbols', a single digit or a percent sign followed by two digits. We propose an alternative syntax that eliminates the symbol that occurs first in the SMILES string (the ring-opening symbol) and replaces the second symbol (the ring closing) by a ring size (using %N for double digits and %(N) for ring sizes greater than 99). Since the successive atoms in a SMILES string occur in depth-first traversal order and thus form a tree, for any atom there is a single path back to the first atom; thus the ring size can be used as a distance along that path to unambiguously identify the location of the ring opening.

For example, the SMILES string c1ccccc1 (representing benzene) is equivalent to the DeepSMILES string cccccc6. This indicates that the final carbon is connected to the atom that occurs six positions before it (in the direct path back to the first atom); this implies a six-membered ring. Note that while a phenyl ring in a larger molecule may sometimes occur in SMILES notation as c2ccccc2, c3ccccc3, etc., it will always occur with the same notation in RP-SMILES, namely cccccc6.

Branches do not cause any ambiguity but illustrate that distance in the SMILES string does not equate to distance from an atom back to the root atom. For example, c1c(F)cccc1 and c1c(cccc1)F are written as cc(F)cccc6 and cc(cccc6)F respectively, but the distance in the SMILES string between the ring-opening and ring-closing atoms is 7 and 6 respectively. One way to think of this is that when counting back to find the ring-opening atom, you should ignore atoms in other branches.

A ring opening symbol in a SMILES string may have an associated explicit or stereo bond preceding it. Since this symbol is eliminated in the transformation to DeepSMILES, the only way to preserve the information is to move it to the ring closing symbol. For example, the SMILES C=2CCC2 is encoded as CCCC=4. Similarly, the SMILES ClC/1=C/CCCCN1 is encoded as ClC1=C/CCCCN\7 (note the change in bond direction).

Stereochemistry is preserved by the transformation to/from DeepSMILES. Ring opening symbols following a tetrahedral centre present some difficulties in this regard as their order determines the stereo configuration but those symbols are eliminated in DeepSMILES. To solve this, we invert the stereochemistry of a tetrahedral centre in those cases where, when reading, we would obtain the wrong stereo configuration. When reading, we place ring closures first and then ring openings are added in order of their corresponding ring closures. For example, the DeepSMILES CC1CCCO[C@@]12CCCCO2 is encoded as CCCCCO[C@@]6CCCCO6, but CC1CCCO[C@@]21CCCCO2 must be encoded with the opposite stereo configuration, as CCCCCO[C@]6CCCCO6.

Transforming parentheses

DeepSMILES avoids the use of paired parentheses by transforming to a syntax related to Reverse Polish Notation (RPN) [17], a parenthesis-free syntax used to describe arithmetic expressions. For example, an equivalent RPN expression for "2 x (4 + 6) + 3 x (7 – 5) =" would be "2 4 6 + x 3 7 5 – x + =". This expression is evaluated by placing each successive digit onto a stack; when an operator is reached, it is applied to the top two items on the stack which are then replaced by the result.

Here we apply a similar notation to transform SMILES strings. The idea behind this approach dates back to 1964 when Hiz [18] and Eisman [19] described postfix line notations that allowed a chemical structure to be written without the use of parentheses. To do so, all atoms are included explicitly (that is, hydrogen atoms are not suppressed in the syntax) and all atoms are assumed (in fact, required) to have their typical valence.

If we wish to preserve in DeepSMILES the hydrogen-suppressed nature of SMILES strings (see Discussion for alternative approaches), it is not possible to achieve an entirely parenthesis free syntax. However, only a single parenthesis is required instead of the two in regular SMILES. The order of atoms in a SMILES string represents a tree traversal of the underlying molecular graph. DeepSMILES preserves the same traversal order but indicates moving to a new branch using parentheses, where the number of parentheses indicates how far back on the current branch the new branch begins. Where no parentheses appear, it means that the next atom is attached directly to the previous atom, while a single parenthesis means that it is instead attached to the second-last atom on the current branch.

An illustrative example is the conversion of the SMILES "C(OC)(SC)F" to the DeepSMILES "COC))SC))F". To convert this back to SMILES, a stack is used, similar to how RPN is interpreted. Each successive atom is connected to the atom on top of the stack (if one is present), and then it itself is placed on top of the stack. Close parentheses remove atoms from the top of the stack, such that the next atom to the added to the stack will attach to an atom 'lower down' on the stack. In this case, C, O and C are added to the stack one after the other, and joined to give COC. Next, the two close parentheses remove the C and then the O from the top of the stack. When the S is added, it attaches

to single C on the stack giving C(OC)S, while the following C gives C(OC)SC. The two final parentheses remove the C and then S from the stack so that the last character F attaches to the original C again.

Additional examples of DeepSMILES strings are shown in Table 1.

SMILES	DeepSMILES
C1CCCC1	CCCCC5
C1CCCCCCCCC1	CCCCCCCCCC%10
C(O)C	CO)C
C(OF)C	COF))C
C(F)(F)C	CF)F)C
C(=O)Cl	C=O)Cl
C(OC(=O)Cl)I	COC=O)Cl))))I
C1CC(OC)CC1	CCCOC))CC5
C1=C/CCCCCC/1	C=C/CCCCCC/8
C\1=C/CCCCCC1	C=C/CCCCCC/8
B(c1ccccc1)(O)O	Bcccccc6))))))O)O
Cn1cccc-2nccc12	Cncccncccc9-5
C1N[C@@]12CO2	CN[C@@]3CO3
[C@@]12(NC1)CO2	[C@@]NC3))CO3
CC1CCCO[C@]21CCCCO2	CCCCCO[C@@]6CCCCO6
CC1CCCO[C@@]12CCCCO2	CCCCCO[C@@]6CCCCO6
NC[C@]12CCCC1C3CC2CC3	NC[C@]CCCC5CCC8CC5
NC[C@]12CCCC2C3CC1CC3	NC[C@@]CCCC5CCC8CC5

Table 1 – Examples of SMILES strings encoded as DeepSMILES, where both the branches and ring closures were transformed.

Results

The ability to accurate encode and decode DeepSMILES from SMILES was tested during development by roundtripping SMILES through DeepSMILES, checking that the canonical SMILES of the original and roundtripped SMILES was identical. The roundtrip test was carried out three times; once transforming both parentheses and ring closures, and then testing each on its own. The datasets used were canonical SMILES (including stereochemistry) generated by the CDK [20], OEChem [21], Open Babel [22], and RDKit [23] for all entries in the ChEMBL 23 database [24]. While canonical SMILES strings are not necessarily the best choice for training a DNN (due to the inherent bias the canonical order introduces), by using strings from multiple programs we cover a variety of traversal orders and ring closure orders (some programs prefer ring closure symbols before ring openings, and *vice versa*).

All of the SMILES strings tested roundtrip without error, although it should be noted that the exact string representation may differ due to different ring closure digits or a rearrangement of ring closure digits, sometimes with an associated inversion of configuration if at a tetrahedral stereocentre. Table 1 shows some performance characteristics for one of the datasets.

Options	Mean % increase in length	Converted per s	
		Encoding	Decoding
Branches	8.2	32000	16000
Rings	-6.4	26000	24000
Both	1.9	26000	17500

Table 2 – Effect of DeepSMILES conversion on string length, and the rate of conversion from SMILES to/from DeepSMILES. The values shown are for canonical SMILES generated by Open Babel for the ChEMBL 23 database.

Discussion

To generate a DeepSMILES string we transform a SMILES string into an alternative syntax that preserves atom order but modifies ring closure symbols and parentheses (see Table 1). The transformation process does not require chemical interpretation of SMILES but only involves string processing of the syntax. As such, it makes certain assumptions about the form of the SMILES string that will be met if the SMILES string was generated by a standard cheminformatics toolkit, but may not be met otherwise. For example, the code assumes that the SMILES was generated as a depth-first traversal of the molecular graph. For example, this assumption is not met by the SMILES string CC(C1)CCCC1 which cannot be encoded as DeepSMILES (at least, while preserving the atom order). The solution is simply to roundtrip such SMILES through such a cheminformatics toolkit to generate the form required.

When transforming ring closure symbols with DeepSMILES, we replace the two ring closure symbols in a pair with a single one, at the location of the second symbol. One may ask why not replace the first symbol instead? The reason is related to the tree structure of a SMILES string; for any node in a tree, there is only a single path leading to it from the root but thereafter the path may split multiple times. Thus, while one can refer to the atom that appeared five bonds earlier (along the path leading to this atom), there may be multiple candidates for the atom that appears five bonds later.

The conversion of two ring closure symbols to a single one addresses the difficulty of generating SMILES strings with matching ring closure digits. However, we believe that this syntax has additional benefits for its application to deep neural nets. Consider that a phenyl ring may be indicated in a SMILES string by a mixture of “c1…..c1”, “c2……c2”, “c3……c3”, and so on, with frequencies that differ along the length of a SMILES string. (As an aside, these frequencies will differ between programs as some reuse ring closure digits as soon as possible, while others may only do so as soon as digit 9 is reached.) In contrast, a phenyl ring will always be present as “c6” in DeepSMILES. Conversely, in SMILES “c1……c1” may indicate a 20-membered ring in one context but a 6-membered ring in another; in DeepSMILES, “c6” always means a 6-membered ring. These differences mean that it should be easier for a machine learning method to learn the correspondence between the string and the underlying structure in the case of DeepSMILES.

The situation is more complex in the case of fused rings. For example, a bicyclic fused ring system consists of three cycles: two smaller ones, and one larger one around the perimeter. This means that, depending on the traversal order of the SMILES string, even if there is a six-membered ring the

digit 6 may not be present as a ring closure symbol in DeepSMILES. Given that the traversal order is arbitrary when writing SMILES, it is worth considering whether there is a preferred traversal order that would make it easier to learn the resulting SMILES or DeepSMILES. For example, if one follows shorter branches first, then the number of parentheses will be reduced. We note that this aspect of SMILES may be less of an issue if a random traversal order is used to generate one or more SMILES for each structure ([25], [26], [27]).

We propose the DeepSMILES syntax as a solution to the problem of invalid SMILES syntax caused by unmatched parentheses and ring closure symbols. While avoiding these problems, the syntax does not entirely eliminate the possibility of invalid strings. Consider the DeepSMILES string "C))C" where there is nothing on the stack for the second parenthesis to pop. Our implementation raises a DecodeError exception when requested to decode this to regular SMILES. An alternative would be to ignore any parenthesis that emptied the stack, thus always yielding a valid SMILES. A similar situation arises for the DeepSMILES string "CCCCC6" where the atom required to complete the ring is not present. Again, our implementation raises a DecodeError exception when decoding, but an alternative might be to use the first atom as the ring opening atom in these cases. It is not clear to the authors whether being tolerant of invalid syntax is a good idea or not; during training one wants the syntax to be learnt as well as possible and so one should not let invalid syntax pass unnoticed but perhaps, when generating structures, 'almost valid' is good enough.

The syntax changes introduced by DeepSMILES represent a subset of all the possible transformations that could be done. For example, consider how fundamental to an atom's identity is the number of heavy atom neighbours, or (almost) equivalently, the number of hydrogens attached. Yet, for atoms in the organic subset and particularly carbon, the relationship between this value and the characters in a SMILES string is non-trivial. However, if we treat the organic subset the same as other atoms, then the number of attached hydrogens would simply be listed for each atom. This may make it easier for DNNs to learn this syntax. In fact, if this were done and in addition we only allow atoms to have a fixed pre-defined valence, it is possible to achieve the parenthesis-free representation of a chemical structure described by Hiz [18] and Eisman [19], which by definition does not allow unusual valences.

With these potential variations in mind, it is worth considering whether if one were to design from scratch a linear notation for molecules for use in Deep Neural Networks, that syntax would be SMILES, DeepSMILES, or indeed a variant of one of the other chemical line notations. What are the features of these strings that aid learning the underlying chemical structure, and what are those that hinder? Perhaps the perfect notation would be one where (1) every string represents a valid molecular structure, (2) there are few duplicate representations, (3) small changes in the string tend to result in small changes to the structure (and *vice versa*) and (4) string size is related to pharmaceutical usefulness or synthetic accessibility.

Finally, it should be noted that despite the name, the proposed syntax is not restricted to applications in deep learning. Any machine-learning method that operates on SMILES strings may benefit from the syntax changes introduced, including methods such as genetic algorithms, n-gram analysis, or grammatical evolution. Even within the field of deep learning, the potential applications

are not restricted to generative models – if the proposed syntax proves to be easier for a DNN to learn, it may be of use for any method that tries to relate the structure to a property.

Conclusion

We have shown that an alternative syntax for SMILES, which we call DeepSMILES, can address aspects of SMILES syntax that cause difficulties for machine-learning methods, particular in the context of generative models. The DeepSMILES syntax describes rings using a single symbol rather than a pair, thus avoiding the problem of unmatched ring closure symbols. The balanced pairs of parentheses required by valid SMILES syntax are replaced by multiple close parentheses that indicate the length of the preceding branch. Roundtripping tests from SMILES to DeepSMILES and back again show that the two notations convey exactly the same information.

We hope that the work described here will make it easier for machine-learning methods to generate syntactically valid SMILES strings, and also that it may prompt the development of new or improved chemical line notations for use in machine learning.

References

1. Gómez-Bombarelli R, Wei JN, Duvenaud D, et al (2018) Automatic chemical design using a data-driven continuous representation of molecules. ACS Central Science 4:268–276 . doi: 10.1021/acscentsci.7b00572

2. Weininger D (1988) SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. J Chem Inf Comput Sci 28:31–36 . doi: 10.1021/ci00057a005

3. Kusner MJ, Paige B, Hernández-Lobato JM (2017) Grammar Variational Autoencoder. arXiv:170301925 [stat]

4. 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 . doi: 10.1021/acscentsci.7b00512

5. Ertl P, Lewis R, Martin E, Polyakov V (2017) In silico generation of novel, drug-like chemical matter using the LSTM neural network. arXiv:171207449 [cs, q-bio]

6. Olivecrona M, Blaschke T, Engkvist O, Chen H (2017) Molecular de-novo design through deep reinforcement learning. Journal of Cheminformatics 9:48 . doi: 10.1186/s13321-017-0235-x

7. OpenSMILES specification. http://opensmiles.org/opensmiles.html. Accessed 13 Sep 2018

8. Yoshikawa N, Terayama K, Honma T, et al (2018) Population-based de novo molecule generation, using grammatical evolution. arXiv:180402134 [physics, q-bio]

9. Dai H, Tian Y, Dai B, et al (2018) Syntax-Directed Variational Autoencoder for Structured Data. arXiv:180208786 [cs]

10. Janz D, van der Westhuizen J, Paige B, et al (2017) Learning a Generative Model for Validity in Complex Discrete Structures. arXiv:171201664 [cs, stat]

11. Heller SR, McNaught A, Pletnev I, et al (2015) InChI, the IUPAC International Chemical Identifier. Journal of Cheminformatics 7:23 . doi: 10.1186/s13321-015-0068-4

12. Kearnes S, McCloskey K, Berndl M, et al (2016) Molecular graph convolutions: moving beyond fingerprints. J Comput Aided Mol Des 30:595–608 . doi: 10.1007/s10822-016-9938-8

13. Simonovsky M, Komodakis N (2018) GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. arXiv:180203480 [cs]

14. Li Y, Vinyals O, Dyer C, et al (2018) Learning Deep Generative Models of Graphs. arXiv:180303324 [cs, stat]

15. You J, Liu B, Ying R, et al (2018) Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. arXiv:180602473 [cs, stat]

16. Jin W, Barzilay R, Jaakkola T (2018) Junction Tree Variational Autoencoder for Molecular Graph Generation. In: International Conference on Machine Learning. pp 2323–2332

17. Burks AW, Warren DW, Wright JB (1954) An Analysis of a Logical Machine Using Parenthesis-Free Notation. Mathematical Tables and Other Aids to Computation 8:53–57 . doi: 10.2307/2001990

18. Hiz H (1964) A Linearization of Chemical Graphs. J Chem Doc 4:173–180 . doi: 10.1021/c160014a015

19. Eisman SH (1964) A Polish-Type Notation for Chemical Structures. J Chem Doc 4:186–190 . doi: 10.1021/c160014a017

20. Willighagen EL, Mayfield JW, Alvarsson J, et al (2017) The Chemistry Development Kit (CDK) v2.0: atom typing, depiction, molecular formulas, and substructure searching. J Cheminf 9:33 . doi: 10.1186/s13321-017-0220-4

21. OEChem, OpenEye Scientific Software, Inc., Santa Fe, NM, USA. http://eyesopen.com/. Accessed 12 Jun 2012

22. O'Boyle NM, Banck M, James CA, et al (2011) Open Babel: An open chemical toolbox. J Cheminf 3:33 . doi: 10.1186/1758-2946-3-33

23. RDKit: Open-source cheminformatics. http://rdkit.org/. Accessed 12 Jun 2012

24. Gaulton A, Hersey A, Nowotka M, et al (2017) The ChEMBL database in 2017. Nucleic Acids Res 45:D945–D954 . doi: 10.1093/nar/gkw1074

25. Jastrzębski S, Leśniak D, Czarnecki WM (2016) Learning to SMILE(S). arXiv:160206289 [cs]

26. Bjerrum EJ, Threlfall R (2017) Molecular Generation with Recurrent Neural Networks (RNNs). arXiv:170504612 [cs, q-bio]

27. Bergwinkl T (2017) Ligand binding affinity prediction using deep learning. In: bergis reptile zoo of software, hardware and ideas. https://www.bergnet.org/2017/02/ligand-binding-deep-learning/index.html. Accessed 13 Sep 2018