Directed reading project report

Students: Qingyuan Feng, Hossein Sharifi Noghabi

May 6, 2017

Abstract

This project is the first step in applying the model proposed in "Automatic chemical design using a data-driven continuous representation of molecules" on a much bigger dataset. Autoencoders were used in the model, which could convert discrete Simplified Molecular-Input Line-Entry System (SMILES) representations of chemical compounds to lower-dimensional latent continuous representations, then convert them back to SMILES strings with the aim of identical reconstruction. The continuous latent representations obtained in the middle of the autoencoders capture some salient chemical features of the molecules. In this project, we used the naive autoencoder applied in the aforementioned paper. A reasonable reconstruction accuracy of 81.25% was achieved, less than the 98.3% or 91.2% in the original paper, which could be possibly attributed to the difference in datasets used and model structure, as well as training time.

1 Introduction

With the rapid development of medical and chemical industries, it is becoming increasingly important to find new drug molecules or chemical compounds that possess some desirable properties. However, due to the difficulty of chemical synthesis and the vast number of candidate molecules, design and evaluation of new drug or chemical compounds are expensive and time-consuming. Therefore, it is rational to find or propose novel molecules with desired properties via reliable and accurate computational methods. To use computational methods, one must overcome challenges such as huge search spaces and the discrete nature of the problem. Recently, autoencoders have been successfully applied in automatic chemical design problem and they offer attractive solutions to the above challenges (Bombarelli, Duvenaud, Miguel, Lobato, & Iparraguirre, 2017).

Autoencoders are a type of Deep Neural Networks (DNN) used for unsupervised learning. It has two components: an encoder, which could learn a latent vector from the input, and a decoder, which could generate an output vector similar to the input from the latent vector. Usually, it is constructed as a feed-forward neural network, and the parameter values are updated via backpropagation algorithm like ordinary DNNs. The latent vector usually has lower dimensionality than the input and output vectors, so it captures some salient features from the input and is often used for dimensionality reduction. The aim of the autoencoder is to generate an output identical to the input, but performs this task by obtaining the vector of informative features and accurately decoding from it (rather than just learning an identity matrix and copying the input).

The model used in our project is presented in figure 1. The inputs and outputs are discrete SMILES string representations of molecules, and the latent vectors are real-valued. There are several advantages with the continuous latent vector of the autoencoder: it allows generating novel chemical compounds by performing basic operations, such as decoding a random vector or perturbing a known chemical structure. Moreover, since the latent space is continuous, gradient-based optimization tools can be utilized to explore and find molecules optimized with respect to some metrics like logP. This project was meant to implement the first step of automatic chemical design, namely the model with a naive autoencoder.

The rest of this report is organized as follows: Section 2 is an introduction to our model, including a brief description of naive autoencoder, then the model used in the main reference of this project is introduced, followed by our implementation of the project and its related details. Section 3 is dedicated to results and discussion, and the last section concludes the report.

2 Model

I. Autoencoders

As mentioned before, an autoencoder is a neural network which is capable of reproducing its input. One might deduce that generating the same input is pointless, however, we are not interested in the output itself; instead, what interests scientific community is its hidden layer h, which contains important information about input x. Usually, autoencoders are under-complete; which means constraints are put on h to have a smaller dimension than x. The hidden layer h could be viewed as an encoding of x, useful in dimensionality reduction problems. Each autoencoder has an encoder part which has the encoding function h = f(x) and a decoder part with function r = g(h) which is responsible for the reconstruction task. The under complete constraint forces the autoencoder to learn the most salient features of the training data. Like other machine learning methods, autoencoders perform the learning process with a loss function which simply penalizes dissimilarities between x and g(f(x)).

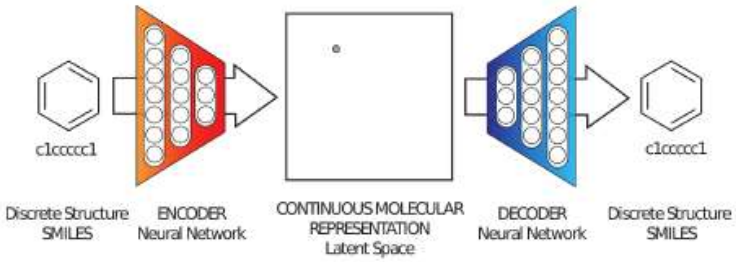

Fig 1. Diagram of the proposed framework for molecular design via autoencoder (Bombarelli, Duvenaud, Miguel, Lobato, & Iparraguirre, 2017)

Some of more well-known types of autoencoder are as follows: regularized autoencoder, sparse autoencoders, denoising autoencoders, stack autoencoders, variational autoencoders, adversarial autoencoders, etc. A detailed description is beyond the scope of this report. Figure 2 illustrates a basic structure of a naive autoencoder.

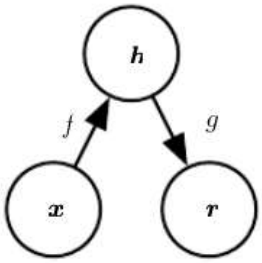

Fig 2. "The general structure of an autoencoder, mapping an input x to an output (called reconstruction) r through an internal representation or code h. The autoencoder has two components: the encoder f (mapping x to h) and the decoder g (mapping h to r)"

II. Automatic chemical design using a data-driven continuous representation of molecules

In the paper, the authors used two types of autoencoders including naive and variational autoencoders to convert discrete SMILES representation of drug-like molecules and organic light-emitting diodes (OLED) to a latent continuous space and convert it back to SMILES format. The reason they used variational autoencoders (VAEs) is that the naive one led to many "dead areas" which do not decode to meaningful molecules. The VAE used in the paper was stochastic on the encoder side and had a penalty term in the decoder network to enforce all of the points in the latent space to correspond to valid decodings. Further, hyperparameters of VAE such as the number of hidden layers, layer sizes, regularization and learning rates were optimized by Bayesian optimization.

They used continuous optimization for mining the generated latent space with the goal of finding new candidate molecules. To correlate the latent representation with target properties, they trained another model to predict molecular properties from the latent vector. By starting from one point and move in the direction most likely to improve the candidate solution and its related property, the new candidate molecules can be decoded from the latent vector obtained. In the paper, logP and QED metrics were selected as examples of optimization.

At the end of the paper, the authors suggested using a graph-based autoencoder which directly outputs molecular graphs could be a promising direction. SMILES representation has a character-by-character nature and fragility in its syntax which can lead to invalid output from the decoder. For more information, please refer to (Bombarelli, Duvenaud, Miguel, Lobato, & Iparraguirre, 2017).

III. Implementation

The implementation section of this report can be divided into two sections as follows:

A. Preprocessing and data preparation

The preprocessing of the data has several steps:

The dataset for this project was acquired from in silico drug design lab at Vancouver Prostate Centre which contains almost 5.5 million drug-like molecules in SMILES representation. This dataset was downloaded from the ZINC15 website. The original data had 6 separate files in .txt format with some irrelevant columns. These 6 files were integrated into one huge file and shuffled with the irrelevant columns removed. The resulting txt file contained only strings of SMILES representation.

After shuffling, the strings were then padded with simple spaces (blanks) on the right to make the SMILES representations of all molecules have a unified length. Since the maximum length in the dataset was 187, all of the other molecules were padded to have length 187.

After the padding step, this padded shuffled dataset was divided into training and test sets. 70% of the data were dedicated to training, the remaining 30% constituted the test set which could evaluate the performance of the model by a new unseen data. Validation set was unnecessary and not used by the original paper, so we didn't include it.

A list of characters appearing in the strings was built. Each character in the string has a corresponding index in the character list, and this index is used to build one-hot vectors. For example, a dataset containing 40,000 strings each with (padded) length 187 and a character list with length 37 would be represented by a (40000, 187, 37)-sized tensor.

B. Naive autoencoder

We implemented the naive autoencoder via Keras with TensorFlow backend.

The model we implemented is a simplified version of (Hodak, 2017), which is, in turn, a reimplementation of (Bombarelli et al, 2017). The reason we did not use Bobarelli et al's original code

was that it was very complicated and difficult to understand. Both of which are available on GitHub. The encoder uses 3 consecutive 1D convolutional layers to convolve the 187 characters, with window sizes (9, 9, 11) respectively. The last convolutional layer is connected to a 435-dimensional fully connected layer, which in turn connects to a 56-dimensional fully connected layer, yielding a 56-dimensional continuous latent vector. The REctified Linear Unit (RELU) activation function was used throughout the encoder network, including the convolutional layers and the layer generating the latent vector. The decoder network takes the latent vector, feeds it into 3 consecutive Gated Recurrent Unit (GRU) layers, all with 501-dimensional outputs. For each timestep (for instance, the 2nd character in a string with length 187), the output from GRU layers goes through a softmax layer to generate a list of probabilities that sum to one (for instance, for the 2nd character with character list ['C', '1'], the output of softmax layer (0.3, 0.7) means that the 2nd character has 30% probability being 'C', and 70% probability being '1'). The whole pipeline is visualized in figure 3.

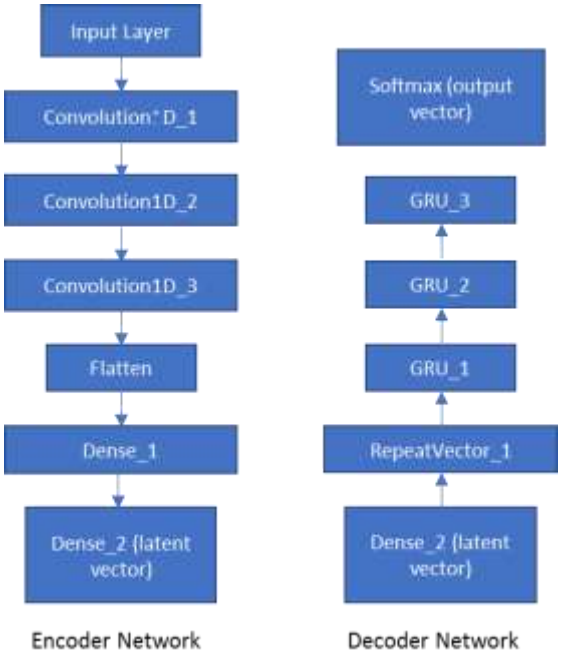

Fig 3. The structure of the autoencoder in our implementation

We briefly introduce the code here. We used argparse library to parse some basic parameters such as the number of epochs, batch size, and dimension of latent space, which were input by the user on the command line. Further, functions such as ModelCheckpoint and ReduceLROnPlateau from callback module in Keras were used in order to save the model after each epoch and decrease the learning rate based on the learning process, respectively. We also implemented our custom callbacks, so that the loss values and accuracy of each batch could be output to a file when the model was being trained.

3 Results and discussion

We used the Helios consortium in Calcul Quebec, a grid in Compute Canada, which has nVidia K20 and K80 GPUs. The code could only run with GPU enabled.

Unlike the experiments done by (Bombarelli et al., 2017), which trained on 250,000 molecules from ZINC12 database and around 100,000 OLED molecules, we trained on around 584,000 molecules from ZINC15 database. Because of time constraint and plateau in performances, we only trained the dataset

for 5 epochs, taking 14 hours in total. Table 1 summarizes the result. The reconstruction accuracies are strict character-by-character accuracies.

Implementation	Training set reconstruction accuracy %	Test set reconstruction accuracy %
Bombarelli et al.	99.1	98.3
This project	80.8	81.3

Table 1. Experiment results comparison

There could be multiple reasons for this inferiority. Our SMILES strings had a maximum length of 187, while in Bombarelli et al's paper, only strings with length no longer than 120 were selected. The hyperparameters used in the convolution layers and GRU layers might not be very suitable for our dataset. We had a larger and different dataset (ZINC15 instead of ZINC12), which may also be a source of the discrepancy. It is also possible, though not quite likely, that there would have been a huge improvement in our results, had we have time for running more episodes.

Besides, the obscurity in (Bombarelli et al., 2017) describing the naïve autoencoder structure probably led us to a suboptimal model structure. The paper did not clearly specify the exact structure of the naïve autoencoder implementation, and there were lots of options in the original code, each corresponding to a different model structure. We picked the structure we thought they used to give 99.1% training set reconstruction accuracy, but it may not be the case. They could have used another model structure which was better than what we implemented.

In general, our result still seemed to be reasonable, though the majority of the SMILES strings we generated might correspond to invalid molecules. In the future, we wish to extend the model using variational autoencoders, try gradient-based optimization and implement the full features of the original research. Afterwards, using the molecular graphs as input (suggested in the original paper) and deducing more molecular properties from the latent representation could be promising research directions. The measure of accuracy could also be modified to take into account of multi-atomic interactions, like functional groups.

4 Conclusion

Following the model in the seminal paper by Bombarelli et al., we implemented the naïve autoencoder taking SMILES representation of molecules as input. Using Keras with TensorFlow backend, we were able to attain a test set reconstruction accuracy of 81.3%, but it was still inferior to the result in the original paper. The difference in model structure, dataset and training epochs may have contributed to this inferiority. Nonetheless, this project could be the first step in a more in-depth research in the field, incorporating computational methods into chemical design, significantly boosting the searching efficiency of new drugs or chemical compounds.

Supplementary Information

The code and relevant information about this project could be found on: https://github.com/simonfqy/molecular-AE-design.

On November 9, 2015, Google Brain released TensorFlow for diverse machine learning tasks. This library is capable of running on multiple CPUs and GPUs and different machines. The name itself is originated from the multidimensional data arrays, called tensors, which neural networks usually operate on. Today, TensorFlow is the most widely used tool in the implementation of deep neural networks in Python.

Another important library for implementation of deep neural networks is Keras. If we assume that with TensorFlow we can build machine learning blocks, Keras is a set of already built blocks that works on top of TensorFlow or Theano. Keras is a high-level neural networks library, written also in Python which is highly modular and extensible.

Statement of contributions:

Hossein was mainly responsible for preprocessing and writing the reports. Qingyuan also did the preprocessing step, following the style of (Hodak, 2017) and revised the reports.

Qingyuan was mainly responsible for implementing the naïve autoencoder.

References

Bombarelli, R. G., Duvenaud, D., Miguel, J., Lobato, H., & Iparraguirre, J. A. (2017, January 6). *Automatic chemical design using a data-driven continuous representation of molecules.* Retrieved from arXiv: https://arxiv.org/abs/1610.02415

Hodak, M. (2017, March 30). *keras-molecules*. Retrieved from github: https://github.com/maxhodak/keras-molecules