École des Mines de Nancy
Université de Lorraine

Deep learning project for molecules clustering and generation

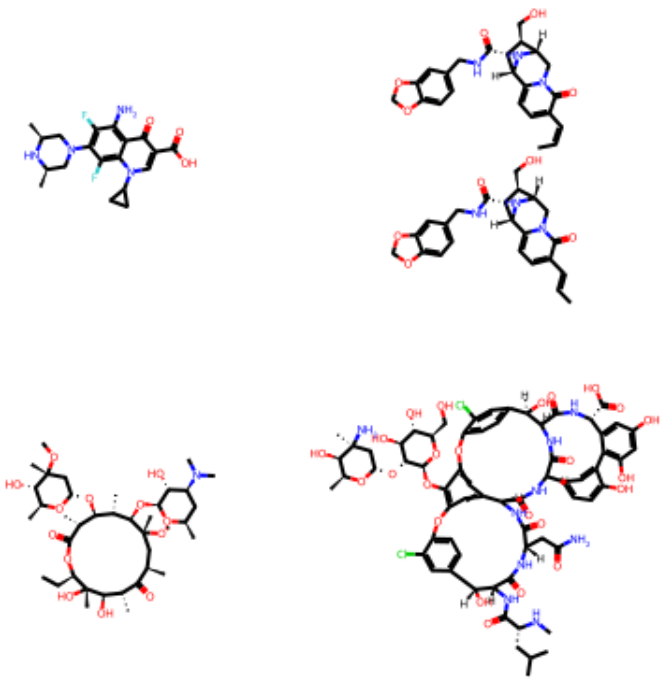
Research Project
Supervised by Parisa Rastin

Martin Jolif

June 2024

Abstract

Machine learning is a relatively new and useful tool to help with several projects about molecules. Indeed, through machine learning and deep learning, we can create tools that are able to predict the properties of molecules, generate new molecules, or classify them automatically according to several pieces of information, like molecular shape, size, chemical composition, functional groups, or other physicochemical properties. In this project, my interest was focused on the clustering of molecules and the generation of new molecules through deep learning. First, the goal of molecular clustering is to identify groups of molecules that share common characteristics or properties. Ultimately, the clusters of molecules obtained will allow researchers to obtain information on the structural relationships between molecules of interest. Secondly, the goal of molecular generation is to propose new molecules to researchers that are similar to the group of data on which the algorithm was trained. These two methods can help in the discovery of new molecules and/or in the prediction of their properties, which can be useful in several fields like agri-food, cosmetics, health, and especially drug discovery [6]. In this project, I relied on several scientific articles and tried to bring some new experiences to them.



Thanks

Before starting this report, I wanted to express my sincere thanks to my supervisor, Parisa RASTIN, for her valuable contribution to the realization of this project. Her support was of great help throughout this year, as she was the one who guided me in learning this vast and complex field. She accompanied me with a lot of patience and teaching, and I am grateful to her.

Contents

Introduction

The goal of the project is to be able to help in the discovery of new molecules that can be useful in several fields like agri-food, cosmetics, health, and especially drug discovery (through machine and deep learning). To do it, my project is separated into two parts: molecule clustering and molecule generation. In each part, I relied on articles and their GitHub codes. In each part, I tried to improve the results by making some little changes in the methods or just by discovering new things. In the rest of this report, I will explain in more detail each theoretical article's methods. For the code part, you can find my code and results on my GitHub profile. The GitHub code of the authors is available in their own article, which is cited in the references.

To be able to use a machine learning algorithm, we will need some numerical representations of molecules that are able to capture information about the molecule, like the structure, the atoms, and the bonds, for example. Molecules can therefore be represented in different ways: SMILES representation, graphs, and images [6]. In this project, to represent molecules, we will use the SMILES representation or the graph representation.

First, the Simplified Molecular-Input Line-Entry System (SMILES) [37] is a specification in the form of a line notation for describing the structure of molecules using short ASCII strings. For example, to represent the water molecule, we use "O" and "C(=O)=O" for the CO_2 molecule. The rules of the SMILES specification are explained here if you are more interested. I used this representation in the first part of my project: the clustering of molecules, and a little bit too in the second part to be able to better visualize the molecules generated by the algorithms.

Secondly, I represented molecules as graphs. A graph is often represented like this: $G = (V, E)$ where V represents a set of nodes and $E = \{e_{i,j} | (i, j) \in V^2, i \neq j\}$ the set of edges. In the case of molecules, V is the set of the atoms numbering (or eventually their one-hot encodings), and E is the matrix of the bonds numbering (or eventually their one-hot encodings, where E is a tensor). I used this representation as input for the generation algorithms.

Chapter 1

Molecules Clustering using SMILES representation

Automatic molecular clustering refers to the process of grouping similar molecules together based on their structural or chemical properties. It is a technique commonly used in various fields, including chemistry, bioinformatics, and drug discovery, to analyze and classify large sets of molecules. The goal of molecular clustering is to identify groups of molecules that share common characteristics or properties. These characteristics may include molecular shape, size, chemical composition, functional groups, or other physicochemical properties. In this project, we will be interested in the automatic learning of a representation space taking into account all of these characteristics by training deep neural networks and in the development of clustering algorithms adapted to these complex data in order to obtain robust and reliable results. Ultimately, the clusters of molecules obtained will allow researchers to obtain information on the structural relationships between molecules of interest, identify recurring motifs, and make predictions about their properties.

To cluster molecules using the SMILES specification, I relied on an article [15] , which used different methods to do it. I reuse their methods and their code, which was available on GitHub, and try to adapt it to find new clustering results on the dataset they used. The article is available here, and the code from the original author is available on his GitHub profile. Moreover, the dataset is available on this website. Like I said before, I relied on their code and modified it to create new methods and find new results.

I reused the methods of another article [16] too. However, I will not explain it in more detail in this report. For more information, see the article and my Github section dedicated to it.

1.1 Method of "Deep clustering of small molecules" article

I will first explain the general method used in the article [15] and explain the different steps from the creation of the features to the model used to obtain clustering results.

1.1.1 Featurization

The method used in this article is to first create features for the molecules using the RDkit library [34]. Different types of features are created for each molecule: some come from the chemical descriptors of the molecule (left part of figure 1.1), and others from the atoms and bonds of the molecule. The atomic and bond information are stored in a matrix for each molecule (middle and right parts of figure 1.1). Then we made a PCA on these two matrices and kept the first principal component.

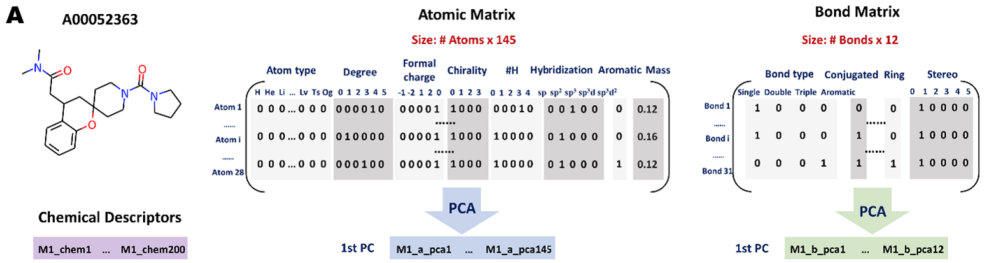


Figure 1.1: First step of the featurization (Image source [15])

The atomic and bond features obtained through PCA are concatenated to obtain a matrix that contains all the molecules. Then another PCA is made, and this time 50 principal components are kept (right part of figure 1.2). At the same time, the matrix of chemical descriptors of all molecules is normalized: each column is centered and reduced (left part of figure 1.2).

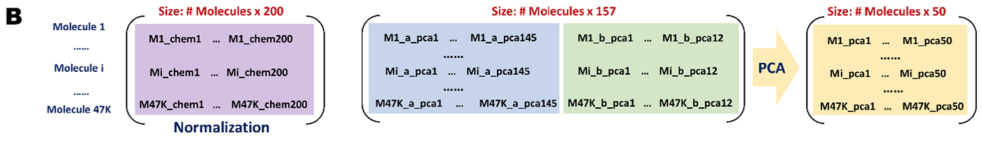


Figure 1.2: Second step of the featurization (Image source [15])

Finally, these two matrixes are concatenated. Then the matrix is filtered; we remove the column with zero-variance (Figure 1.3).

Figure 1.3: Final step of the featurization (Image source [15])

The final matrix has a size of Number of Molecules∗251 in my code because I have a Chemical Descriptors Matrix of size Number of Molecules∗208. Now that we have completed our featurization, we can apply different methods and algorithms to this final matrix to cluster the molecules in the dataset.

1.1.2 Visualization of the model and its results

After having obtained the "cleaned" feature matrix, we can apply a lot of clustering algorithms to it. However, this is a matrix of size: Number of Molecules∗251. That's why it's maybe more useful first to reduce the dimension of the features and then to apply clustering algorithms (further information on that is in Section 1.5). I had done it with and without reducing the dimensions of the features to compare the results.

Models on the entire space

In this subsection, the algorithms are applied to the "cleaned" matrix obtained at the end of the subsection 1.1.1 of size Number of Molecules∗251.

Algorithm 1 Models on the entire space

Need dataset D which contains SMILES representation of molecules

1: We apply the featurization explained in section 1.1.1 on the dataset D to obtain the matrix M.
2: We apply a clustering algorithm to this matrix M to obtain a clustering

Kmeans The number of clusters K is a hyper-parameter that has to be chosen by the user (see 1.3.1). So we can choose it through the silhouette coefficient (see 1.4.2).

The figure 1.4 shows us that a good choice will be to take $K = 130$. After running the Kmeans algorithm for $K = 130$ we obtain the results of the table 1.1.

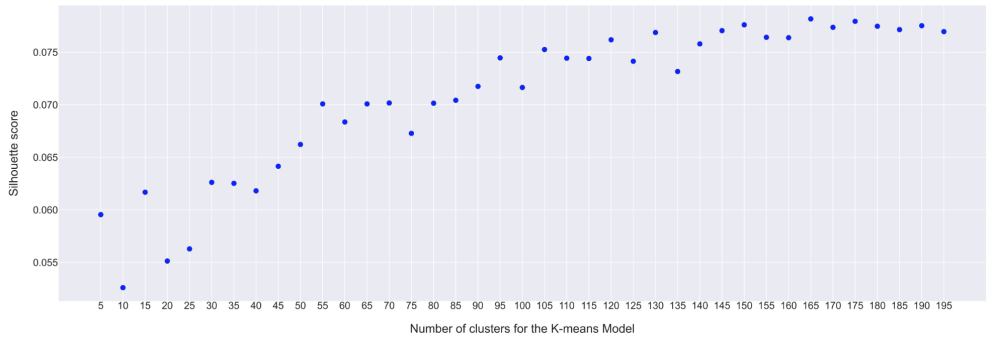


Figure 1.4: Representation of the silhouette coefficient as a function of the number of clusters

Calinski-Harabasz Index ↑	Silhouette coefficient ↑	Davies-Bouldin Index ↓
533.264	0.076	1.949

Table 1.1: Results for Kmeans (K=130)

Models on the latent space after passing a VAE

A Variational Auto Encoder (VAE) is a method of dimension feature reduction (see 1.5.3 for further explanations). Like I said before, this method is used to simplify the computational cost and to save the most information. After passing the VAE, the "cleaned" matrix of size Number of Molecules∗251 is reduced to a "latent" matrix of size Number of Molecules∗32 (32 is the dimension of the latent space of the VAE). Then, we can apply different clustering algorithms to this new "latent" matrix in the latent space. Once the clusters have been chosen by the algorithm, we can decode the "latent" matrix to obtain a matrix very similar to the original one ("cleaned" matrix), compute the three indexes that we use in this project (see 1.4) and compare the results to the model that used the entire space (1.1.2).

Algorithm 2 Models in the latent space

Need dataset D which contains SMILES representation of molecules

1: We apply the featurization explained in section 1.1.1 on the dataset D to obtain the matrix M.
2: We apply the VAE with the matrix M as input, to obtain another matrix Z.
3: We apply a clustering algorithm to this matrix Z to obtain clustering in the latent space.

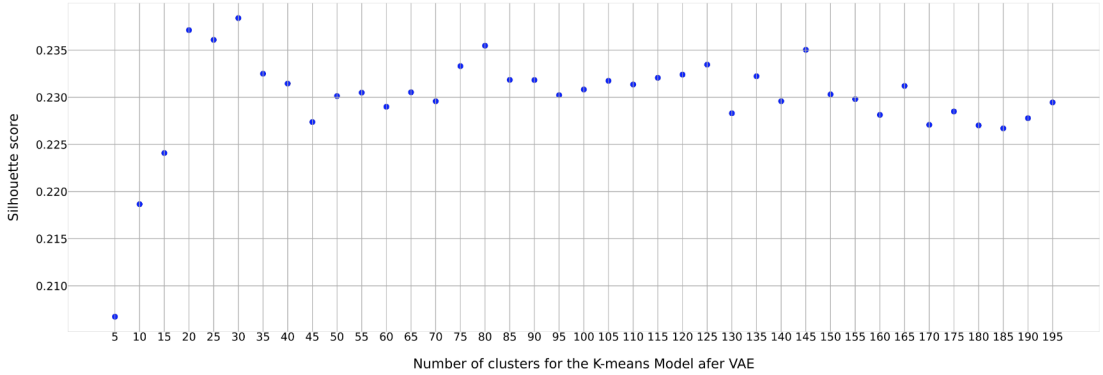


Figure 1.5: Representation of the silhouette coefficient as a function of the number of clusters K

For example, for a latent space of dimension 32, and a number of clusters of 30 (see figure 1.5), for the Kmeans algorithm, I obtained the following results :

Calinski-Harabasz Index ↑	Silhouette coefficient ↑	Davies-Bouldin Index ↓
6452.638	0.240	1.091

Table 1.2: Results for VAE (32) + Kmeans (K=30)

1.1.3 Representation of the clusters found by a model

To verify the coherence of the clusters found by one of the previously modeled models, a means will be to speak with a molecular specialist, show him molecules that are in the same cluster, and ask him if he found it coherent or not. So, I implemented a little code that asks the user for a cluster digit, and the program returns the classic representation of five random molecules that belong to this cluster. For example, the program returns this (see figure 1.6) when we choose the Kmeans algorithm after a VAE and the cluster number 0.

Thanks to this program, we can better understand the clustering method and modify the algorithm if there is incoherence. We can change the algorithm or the dimension reduction method, or just change hyper-parameters like the number of clusters K for Kmeans.

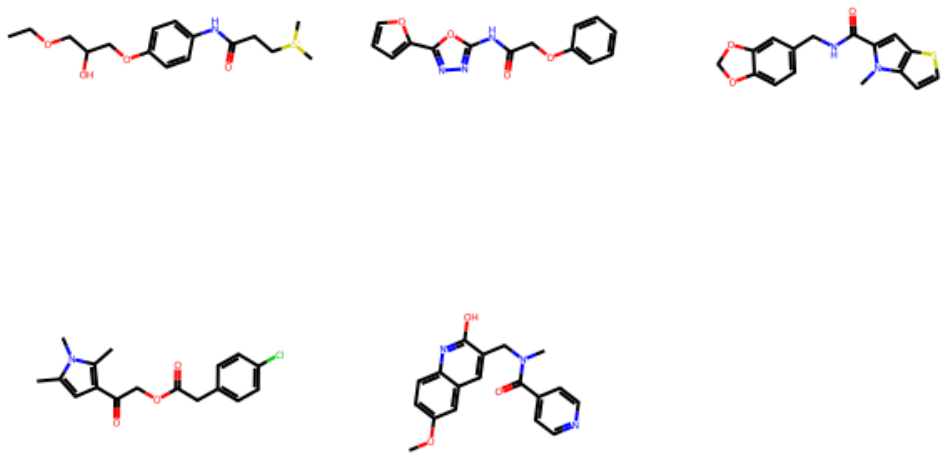


Figure 1.6: Image returned by the program for cluster number 0 (VAE + Kmeans)

1.2 New models and results

In this section, I will explain the new methods that I implemented to find new clustering results and compare them together with the results from the original article.

In the original article [15], the authors only used VAE and Kmeans. To try something new and see if it could give better results, I tried VAE + different algorithms like BIRCH, DBSCAN, OPTICS, bisecting Kmeans, and Meanshift.

Here are some results that I obtained for a latent space of dimension 32 (see table 1.3, 1.4, 1.5, 1.6, 1.7, 1.8)

We observe through the different results that passing through the VAE is helping to obtain clusters with better metrics results. However, it's still difficult to compare the results between algorithms without and with VAE because the metrics are then computed in two different spaces (input and latent space).

We observe that it's difficult to choose the best algorithm that gives good metric results. Moreover, the results often depend on the number of clusters. To choose our final algorithm, we have to make a compromise between the number of clusters that we want and an algorithm that gives good metrics results. It's not easy!

Number of Clusters	CH ↑	SC ↑	DB ↓
25	**6747.221**	0.239	1.100
26	6688.481	0.236	1.103
27	6616.331	0.236	1.125
28	6579.824	0.239	1.105
29	6503.428	0.239	**1.085**
30	6452.638	**0.240**	1.091
31	6353.183	0.234	1.130
35	6100.477	0.231	1.139
40	5901.286	0.231	1.169
50	5454.641	0.232	1.153
80	4620.676	0.234	1.14

Table 1.3: Results for VAE (32) + Kmeans

Number of Clusters	CH ↑	SC ↑	DB ↓
10	3966.486	0.133	1.386
15	**4262.498**	0.129	1.334
20	3832.337	0.145	1.248
25	3809.872	**0.165**	**1.140**
30	3451.681	0.151	**1.140**
35	3589.347	0.149	1.159
40	3260.281	0.147	1.146
45	3232.431	0.154	1.168
50	3080.691	0.149	1.177

Table 1.4: Results for VAE (32) + BIRCH (threshold = 0.5, branching-factor=50)

Number of Clusters	Epsilon	CH ↑	SC ↑	DB ↓
94 + outlier points	0.3	170.611	-0.344	1.602
19 + outlier points	0.5	100.915	-0.115	1.802
15 + outlier points	0.6	112.517	0.094	1.631
12 + outlier points	0.7	137.627	0.232	1.556
6 + outlier points	0.8	**198.313**	0.348	1.603
9 + outlier points	1.2	148.970	0.443	1.608
4 + outlier points	1.5	128.754	0.486	1.241
2 + outlier points	2	132.724	**0.575**	**0.968**

Table 1.5: Results for VAE (32) + DBSCAN

Number of Clusters	min samples	CH ↑	SC ↑	DB ↓
1493 + outlier points	5	11.185	-0.532	1.236
280 + outlier points	10	25.169	-0.636	1.116
107 + outlier points	15	47.644	-0.598	1.012
23 + outlier points	25	113.761	-0.355	**0.928**
5 + outlier points	50	**121.111**	**-0.150**	0.936

Table 1.6: Results for VAE (32) + OPTICS

Number of Clusters	CH ↑	SC ↑	DB ↓
5	**8112.950**	0.161	1.621
10	6677.805	0.140	1.561
15	6371.956	**0.163**	1.469
20	5715.197	0.156	1.424
25	5190.460	0.144	1.437
30	4971.206	0.153	**1.390**
35	4753.198	0.152	1.455
40	4671.040	0.157	1.410
45	4443.872	0.154	1.419
50	4242.036	0.153	1.431
60	3952.478	0.150	1.408
70	3695.272	0.146	1.410
80	3516.554	0.144	1.399

Table 1.7: Results for VAE (32) + Bisecting Kmeans

Number of Clusters	Bandwidth	CH ↑	SC ↑	DB ↓
22	None	151.947	0.233	0.847
8	3	213.409	0.484	1.329

Table 1.8: Results for VAE (32) + Meanshift

1.3 Clustering algorithms

In this section, I will explain in more detail how some of the clustering algorithms that I used in this clustering project work.

1.3.1 Kmeans

This algorithm is very famous because it gives very good results in a short amount of computational time. The hyperparameter that the user must choose for this algorithm is the number of clusters write K in this article. The goal of this

algorithm is to minimize this function, often called inertia.

$$I(C_1, \ldots, C_K) = \sum_{j=1}^{K} \sum_{x \in C_j} \|x - c_j\|_2^2 \tag{1.1}$$

where $c_j = \frac{1}{|C_j|} \sum_{x \in C_j} x$ is the centroid of the cluster j.

A method to implement this idea would be to first initialize the centroid clusters (the value of the c_j). A simple idea is to set the first K centroid clusters to K samples from the dataset. K-means involves looping between the two other steps after startup. Each sample is assigned to the closest centroid in the first stage. By taking the mean value of all the samples allocated to each previous centroid, new centroids are created in the second step. The remaining two steps of the algorithm are repeated until the difference between the old and new centroids is smaller than a threshold. In other words, it repeats until the centroids do not move significantly.

See this course [33] for further information and the documentation of scikit-learn [31].

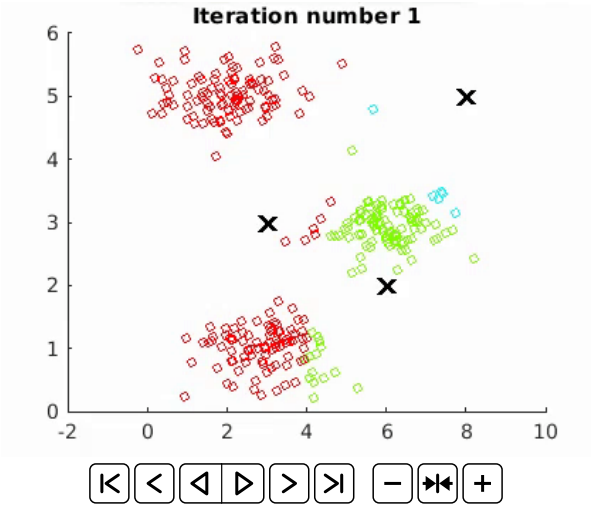


Image source [5]

1.3.2 DBSCAN

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is an algorithm based on density. The clustering algorithm is linked to the estimation of the probability density: a cluster will be identified like a set of samples in a region of high density. The hyperparameters that the user must choose for the DBSCAN algorithm are ε (a non-negative reel) and a natural integer N (or *minsamples* in the scikit-learn documentation).

A sample is considered a core sample in the dataset if it has at least N neighbors within a distance of ε : x is a core sample if there are at least N samples in $B(x, \varepsilon) = \{y \in E \mid \|x - y\|_2 \leq \varepsilon\}$. It tells us that the core sample is situated in a dense area of the dataset. Moreover, Any sample that is not a core sample, and is at least ε in distance from any core sample, is considered an outlier by the algorithm. A cluster is a set of core samples that can be built by recursively taking a core sample, finding all of its neighbors that are core samples, finding all of their neighbors that are core samples, and so on (see the gif below). Finally, a cluster also has a set of non-core samples, which are samples that are neighbors of a core sample in the cluster but are not themselves core samples. Intuitively, these samples are on the border of a cluster. These are the samples that have a core sample at a distance of less than epsilon but are not core samples.

An advantage of this algorithm is that we don't need to choose the number of clusters K before running it. Moreover, this algorithm can find and classify some samples as outlier points. A drawback of this algorithm is that the hyperparameters ε and N are uniform on the whole dataset. A consequence is that it is difficult to identify clusters with different densities.

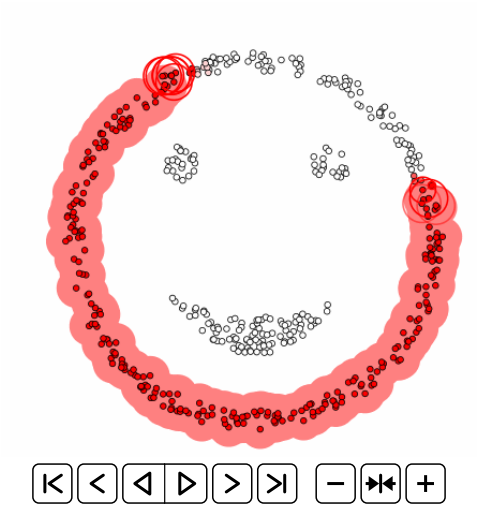


Image source [13]

1.3.3 Mean-shift

This algorithm is explained in more detail in this article [4]. This algorithm is based on a centroid, a little bit like the Kmeans algorithm: the centroid is updated at each iteration of the algorithm. The centroid x is updated like this between iterations t and $t + 1$:

$$x^{t+1} = x^t + m(x^t) \tag{1.2}$$

First, the algorithm computes the mean shift vector:

$$m(x) = \frac{1}{|N(x)|} \sum_{x_j \in N(x)} x_j - x \tag{1.3}$$

where $N(x)$ is defined as the neighborhood of samples within a given distance

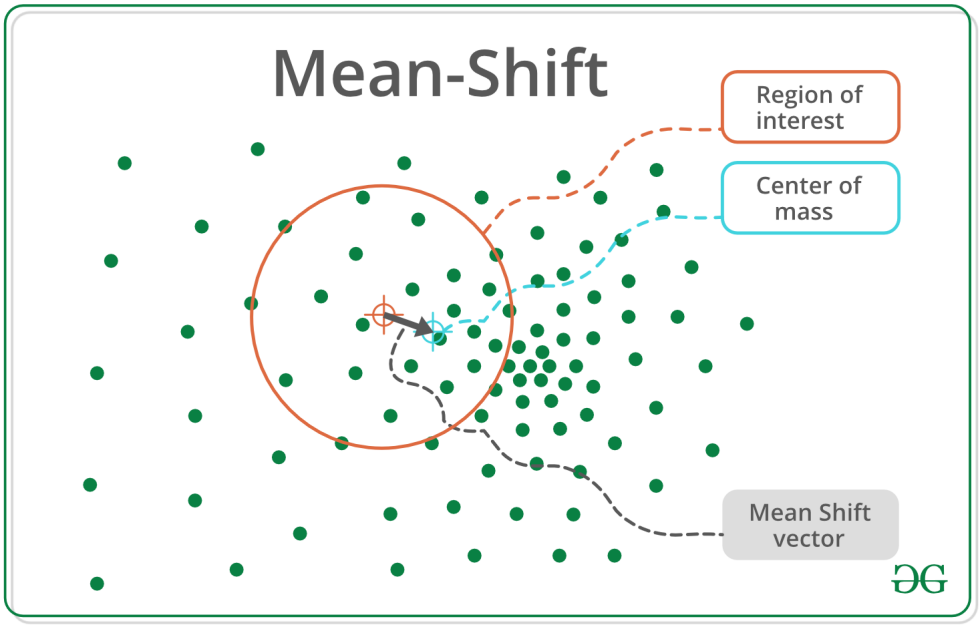


Figure 1.7: Mean Shift vector (Image source [23])

around x. So on the figure 1.7, $N(x^t)$ is the area in the orange circle and $m(x^t)$ is the black vector. After the step t, all the points in the orange circle will be moved with the mean shift vector. At the end, when between two steps, all the points are not moving far anymore, we can see clusters (see the gif) :

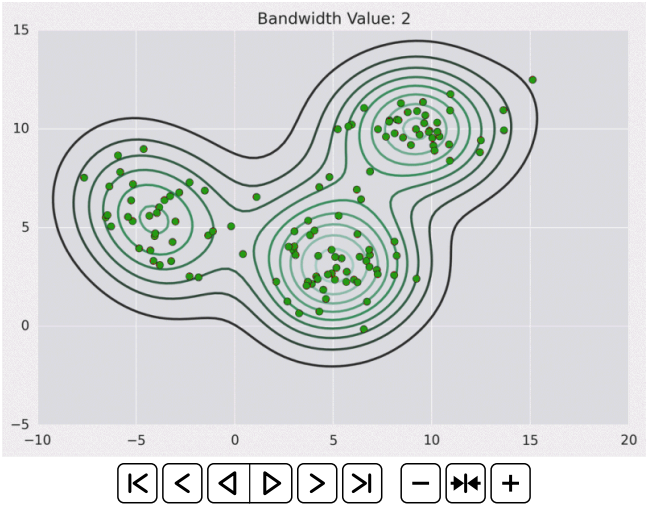


Image source : [23]

In the Scikit-learn implementation, $m(x)$ is computed like this :

$$m(x) = \frac{\sum_{x_j \in N(x)} K(x_j - x) x_j}{\sum_{x_j \in N(x)} K(x_j - x)} - x \tag{1.4}$$

where K is a kernel. $K(x)$ is equal to 1 if x is small enough and equal to 0 otherwise. The goal is that $K(x - y)$ indicates if x and y are close. For example, $K(x)$ can be equal to $e^{-\frac{\|x\|^2}{2}}$ for $c > 0$. The algorithm updates the values of the centroids until it converges. Another example of K is $K(x) = W(\frac{x}{h})$, where W is defined on equation 1.7. The mean shift vector can therefore be defined like this:

$$m(x) = \frac{h^2 \nabla p_W(x)}{p_W(x)} = \lambda(x) \nabla p_W(x) \tag{1.5}$$

Where the estimation of the density of the point x in the sample space is given by:

$$p_W(x) = \frac{1}{|E| h^n} \sum_{y \in E} W(\frac{x - y}{h}) \tag{1.6}$$

where $N = |E|$ is the number of samples, n is the dimension of the space associated with the samples, and h is the bandwidth parameter of this algorithm. Moreover, we suppose that:

$$W(x) = W_0 \exp(-\frac{\|x\|_2^2}{2}) \tag{1.7}$$

such that $\int W = 1$

Like the DBSCAN algorithm, the algorithm automatically chooses the number of clusters.

1.4 Clustering metrics

In this section, I will explain the three metrics that I used in my clustering project to compare the results of different methods. To compute these scores, the clusters must be made and the molecules labeled. We assume that there are K clusters and that the space E of dimension n is clustered like this.

$$E = \bigsqcup_{k=1}^{K} C_k \tag{1.8}$$

where $C_k = \{x \in E \mid x \text{ is in cluster } k\}$. So we have the following equality :

$$|E| = \sum_{k=1}^{K} |C_k| \tag{1.9}$$

Moreover, $\|\cdot\|_2$ represent the euclidean norm : for $x \in E$, $\|x\|_2 = \sum_{k=1}^{n} |x_k|^2$.

1.4.1 Calinski-Harabasz Index

This index is explained in more detail in this article [3].

If we write c_q the center of the cluster q : $c_q = \frac{1}{|C_q|} \sum_{x \in C_q} x$, $n_q = |C_q|$ the number of samples in cluster q and c_E the center of E : $c_E = \frac{1}{|E|} \sum_{x \in E} x$. Then, we can compute W and B the within cluster dispersion and the between group dispersion coefficient:

$$W = \sum_{k=1}^{K} \sum_{x \in C_k} \|x - c_k\|_2^2 \tag{1.10}$$

$$B = \sum_{k=1}^{K} n_k \|c_k - c_E\|_2^2 \tag{1.11}$$

Then the Calinski-Harabasz Index is computed like this according to equation 1.10 and 1.11 :

$$CH = \frac{B}{W} \frac{|E| - K}{K - 1} \tag{1.12}$$

We noticed that the score is higher for clusters that are well separated : it's related to the standard notion of cluster that we have. The score is fast computed (much faster than the silhouette coefficient 1.4.2). However, the score is generally higher for clusters obtained through density algorithms like DBSCAN (see 1.3.2).

1.4.2 Silhouette coefficient

This index is explained in more detail in this article [30]. I will first explain how to compute the silhouette score of one sample. If we note x, the coordinate vector of the molecules in the space E. We compute a, the average distance between x and all other samples that belong to the same cluster as x, which is noted as cluster i. We compute b too, the average distance between x and all other samples that belong to the nearest cluster of x, which is noted as cluster j. Then we obtain the equations 1.13 and 1.14:

$$a(x) = \frac{1}{|C_i|} \sum_{y \in C_i} \|x - y\|_2 \tag{1.13}$$

$$b(x) = \frac{1}{|C_j|} \sum_{y \in C_j} \|x - y\|_2 \tag{1.14}$$

Then, the silhouette score of x is

$$sc(x) = \frac{b(x) - a(x)}{\max(a(x), b(x))} \tag{1.15}$$

and then, the silhouette score of the complete partition according to the above explanation is :

$$SC = \frac{1}{|E|} \sum_{k=1}^{K} \sum_{x \in C_k} sc(x) \tag{1.16}$$

Thanks to the equations 1.15, 1.16, and 1.9, we notice that $SC \in [-1, 1]$. Moreover, the higher the score, the better the space E is clustered. Indeed, when we have a score near 1, it means that the clusters are well separated and tight. However, if we have a score near 0, it means that the clusters are overlapping with each other's. Finally, a score near -1 corresponds to incorrect clustering. It's related to the standard notion of cluster that we have. However, the silhouette score is generally higher for clusters obtained through density algorithms like DBSCAN (see 1.3.2) and takes a relatively long time to be computed, especially for big clusters.

1.4.3 Davies-Bouldin Index

This index is explained in more detail in this article [7]. If we write s_i the average distance between each point of cluster i and the centroid of that cluster c_i : $c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$, then we have:

$$s_i = \frac{1}{|C_i|} \sum_{x \in C_i} \|x - c_i\|_2 \tag{1.17}$$

Moreover, we write d_{ij} the distance between cluster centroids i and j :

$$d_{ij} = \|c_i - c_j\|_2 \tag{1.18}$$

Thanks to equation 1.17 and 1.18, we can construct R_{ij} such that it is non negative and symmetric :

$$R_{ij} = \frac{s_i + s_j}{d_{ij}} \tag{1.19}$$

Then, the Davies-Bouldin Index is defined as :

$$DB = \frac{1}{K} \sum_{i=1}^{K} \max_{j \neq i} R_{ij} \tag{1.20}$$

We then observe that a lower Davies-Bouldin index relates to a model with better separations between clusters. As we said before, zero is the lowest value of this metric. The computation of this index is much simpler and quicker than the silhouette score. However, the score of this metric is often higher for convex clusters, like those obtained with the DBSCAN algorithm, for example.

1.5 Dimension reduction methods

In this section, I will explain the different dimension reduction methods that I used for this clustering project. The goal of the different dimension reduction method is to use the "cleaned" matrix of size Number of Molecules$*$251 obtained in the featurization part of the article (section 1.1.1) to create a new features matrix that has a lower number of features (columns) and that contains almost all of the information contained in the original "cleaned" matrix. The aim is to counteract the curse of dimension and computation cost while maintaining a maximum of variance.

1.5.1 PCA

The Principal Component Analysis (PCA) is a linear-dimensional reduction method. We write X as a matrix of size $n * p$ where each column has been centered : each column has a mean of zero.

First proof

The transformation to obtain l new features ($l < p$) is defined as a set of size l of p-dimensional vectors of coefficients such that for $(i, k) \in [\![1, n]\!] \times [\![1, l]\!]$:

$$t_{k(i)} = \langle x_{(i)} \ , \ w_{(k)} \rangle \tag{1.21}$$

where $w_{(k)} = (w_1, \ldots, w_p)_{(k)}$ are the weights that map each row vector $x_{(i)}$ of X to a new vector $t_{(i)} = (t_1, \ldots, t_l)_{(i)}$ and $t_{k(i)}$ is the k-th coordinates of the vector $t_{(i)}$. Moreover the vector w is constrained to be a unit vector. In order to maximize the variance, the first vector $w_{(1)}$ must satisfy this condition :

$$w_{(1)} = \arg \max_{\substack{\|w\|=1 \\ w \in \mathbb{R}^p \setminus \{0\}}} \sum_{i=1}^{n} (t_1)^2_{(i)} = \arg \max_{\substack{\|w\|=1 \\ w \in \mathbb{R}^p \setminus \{0\}}} \sum_{i=1}^{n} \langle x_{(i)} \ , \ w \rangle^2 = \arg \max_{\substack{\|w\|=1 \\ w \in \mathbb{R}^p \setminus \{0\}}} \|Xw\|^2 \tag{1.22}$$

Indeed, we want the inertia of the samples projected into the principal component to be as high as possible. So, as w has been defined as a unit vector ($\|w\| = 1$), we obtain:

$$w_{(1)} = \arg \max_{\substack{\|w\|=1 \\ w \in \mathbb{R}^p \setminus \{0\}}} w^\top X^\top X w = \arg \max_{w \in \mathbb{R}^p \setminus \{0\}} \frac{w^\top X^\top X w}{w^\top w} \tag{1.23}$$

The function to maximize is well known as the Rayleigh quotient. For a symmetric positive semi-definite matrix such as $X^\top X$, a standard result is that the maximizer of the Raygleigh function is the eigenvector (wrote e_n) associated with the eigenvalue λ_n ($\lambda_0 \leq \cdots \leq \lambda_n$ where the λ_i are the eigenvalues of the matrix $X^\top X$). So, we obtain $w_{(1)} = e_n$. Then the first principal component, $t_{(1)}$, can be computed thanks to equation 1.21.

For the further components, the k-th component can be found through the first k-1 components of X. If we defined $\hat{X}_k$ as :

$$\hat{X}_k = X - \sum_{s=1}^{k-1} X w_{(s)} w_{(s)}^\top \tag{1.24}$$

Then, if we define $V_k = Vect(\{w_{(1)}, \cdots, w_{(k-1)}\})$,

$$w_{(k)} = \arg \max_{\substack{\|w\|=1 \\ w \in \mathbb{R}^p \setminus V_k}} \|\hat{X}_k w\|^2 = \arg \max_{\substack{\|w\|=1 \\ w \in \mathbb{R}^p \setminus V_k}} \|Xw\|^2 \tag{1.25}$$

because $\hat{X}_k^\top \hat{X}_k$ is proportional to $X^\top X$. Then, as $w \in \mathbb{R}^p \setminus V_k$, $w_{(k)} = e_{n-k+1}$. The full principal components decomposition can therefore given by :

$$T = XW \tag{1.26}$$

where $W \in \mathcal{M}_{p,p}(\mathbb{R})$ is a matrix whose columns form an orthonormal basis and are the eigenvectors of the matrix $X^\top X$. Likewise, $T \in \mathcal{M}_{n,p}(\mathbb{R})$ and each principal component is stored in each row of the matrix T.

Second Proof

Another method to find principal components is to use the Singular Value Decomposition (SVD) of a matrix. Thanks to the SVD decomposition, there exists $\Sigma \in \mathcal{M}_{n,p}(\mathbb{R})$ a rectangular diagonal matrix of positive numbers wrote σ_k, $U \in \mathcal{M}_{n,n}(\mathbb{R})$ a matrix where the columns form an orthonormal basis and $W \in \mathcal{M}_{p,p}(\mathbb{R})$ a matrix where the columns form an orthonormal basis too, such that:

$$X = U\Sigma W^\top \tag{1.27}$$

Then, the $X^\top X$ matrix can be written like this because $U \in \mathcal{O}_{n,n}(\mathbb{R})$:

$$X^\top X = W\Sigma^\top U^\top U\Sigma W^\top = W\Sigma^\top \Sigma W^\top \tag{1.28}$$

Then, we deduced from equation 1.28 that $\sigma_k = \sqrt{\lambda_k}$ where the λ_k are the eigenvalues of the matrix $X^\top X$. So, we obtain thanks to $W \in \mathcal{O}_{p,p}(\mathbb{R})$ and equation 1.27 that:

$$T = XW = U\Sigma W^\top W = U\Sigma \tag{1.29}$$

This is the polar decomposition of T. Efficient algorithms exist to compute the SVD, in the machine learning library scikit-learn, the PCA function uses the SVD decomposition.

1.5.2 Autoencoder

Autoencoding is the process of encoding data automatically. An Autoencoder (AE) is made of an encoder (the blue part) and a decoder (the yellow part) in the figure 1.8.

The role of the encoder is to "encode" the observation x through his features into a lower-dimensional representation z. This is done by passing several neural network layers. The lower-dimensional representation is the "encoded data", which is included in the latent space. The decoder is learning back from the latent space to reconstruct the observation, the result obtained from the decoder is written as $\hat{x}$, like in figure 1.9.

Therefore, the reconstruction loss function corresponding to this problem that we want to minimize is $\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2 = \|x - d(e(x))\|^2$ where e and d cooresponds to the encoder and decoder. However, the dimensionality of the latent space has a huge impact on the quality of the reconstruction: the lower the dimension, the lower the quality of the reconstruction is going to be. However, the lower the dimension, the lower the storage is going to be.

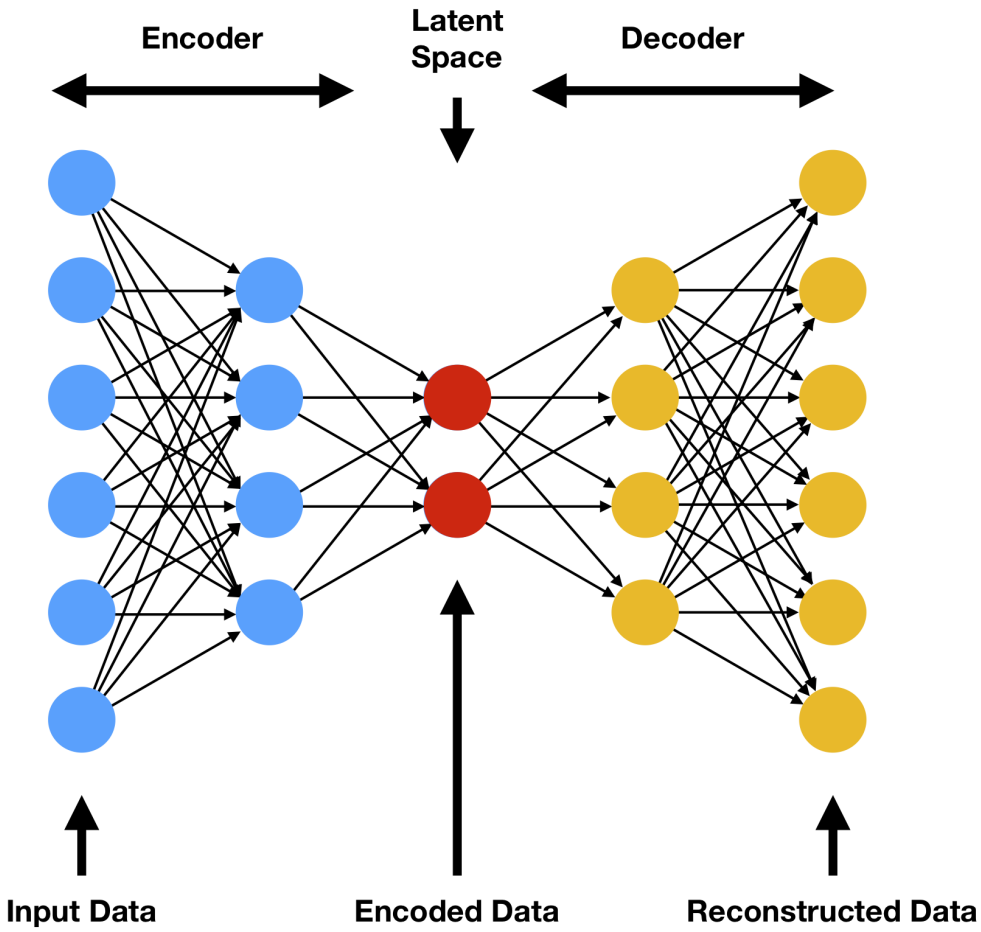


Figure 1.8: Structure of an Autoencoder (Image source [12])

1.5.3 Variational Autoencoder

The principle of a Variational Autoencoder (VAE) is the same as that of an AE. In contrast to an AE, the VAE isn't deterministic because of the weights of the neural networks. Indeed, the VAE brings some randomness to the AE process. Instead of learning the deterministic latent variables, for each latent variable we replace it by the learning of a couple (μ, σ) which represent the mean and the standard deviation of a normal distribution (μ is a vector and σ a matrix). See the figure 1.10.

This time, the function to minimize is:

$$\mathcal{L} = \text{reconstruction loss} + \text{regularization term}$$

The reconstruction loss is the same as the AE: $\|x - \hat{x}\|^2 = \|x - d(e(x))\|^2$. The

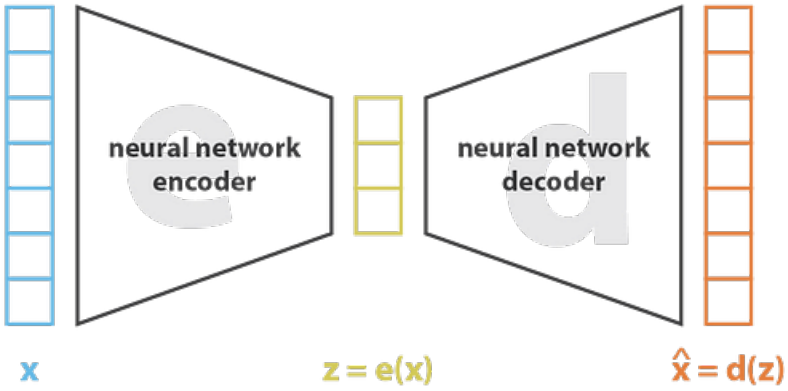


Figure 1.9: Structure of an Autoencoder (Image source [29])

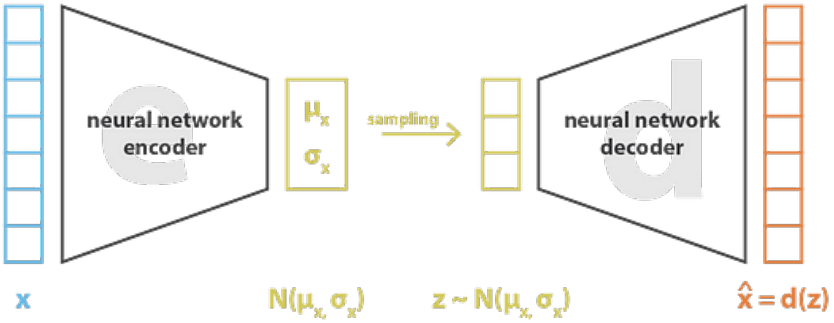


Figure 1.10: Structure of a Variational Autoencoder (Image source [29])

regularization term represents the distance between our prior and the latent distribution found by the encoder. It can be represented like this: where $p(z)$ is a prior on the latent distribution. It's an initial hypothesis on how the latent space may look, and $D(\cdot\|\cdot)$ is a distance. A common choice for the prior is to use a standard normal Gaussian distribution. The regularization term is then,

$$D(p_\phi(z|x) \parallel p(z)) \tag{1.30}$$

where $p_\phi(z|x)$ represents what the encoder computes (see figure 1.11). When we make the choice that the prior is a standard normal Gaussian distribution, we obtain the form of $D(p_\phi(z|x) \parallel p(z))$, which is called Kullback-Leibler-Divergence

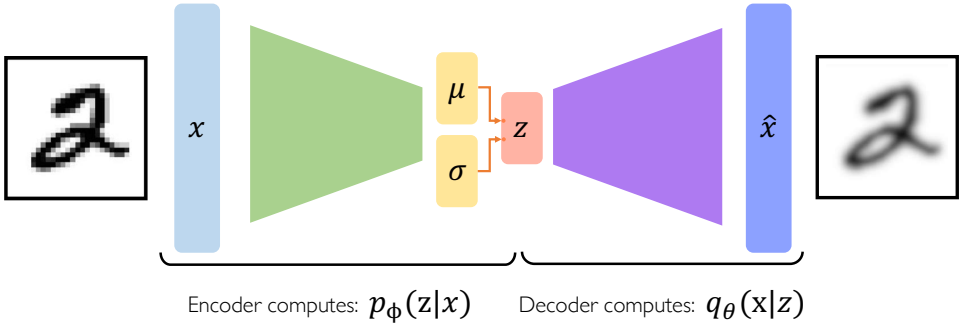


Figure 1.11: Structure of a Variational Autoencoder (Image source [1, 22])

(KL-Divergence is a measure of dissimilarity between probability distributions):

$$D(p_\phi(z|x) \parallel p(z)) = -\frac{1}{2}\sum_{j=0}^{k-1}(\sigma_j + {\mu_j}^2 - 1 - \log \sigma_j) \tag{1.31}$$

where $z \sim \mathcal{N}(\mu, \sigma^2)$.

The goal of regularization is that points that are close in the latent space will be similarly and meaningfully decoded. It would not be the case if we had not put in place the regularization: some close points in the latent space would not be similarly decoded, it's a problem for the decoder and interpretation.

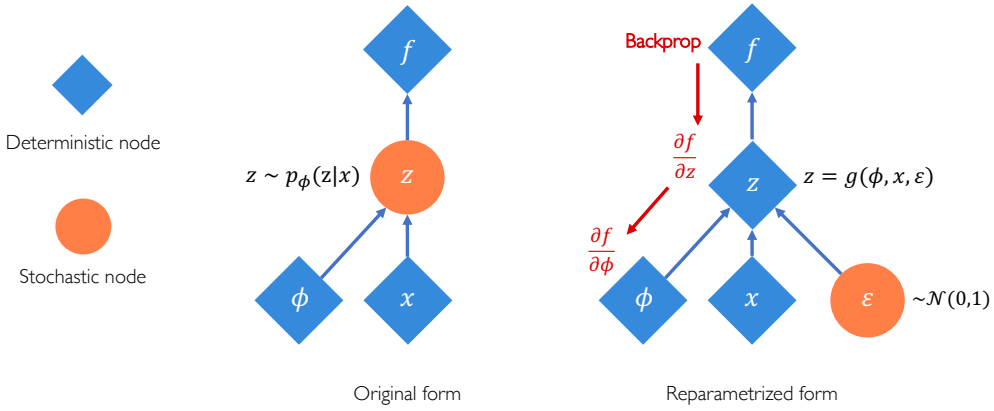


Figure 1.12: Reparametrization Trick of a VAE (Image source [1, 22])

However, the problem for the decoder part is that we cannot backpropagate gradients through sampling layers. The idea to solve this problem is to reparameterize the sampling layer (see figure 1.12). Instead of sampling the vector $z \sim \mathcal{N}(\mu, \sigma)$ like that, we sample it like that :

$$z = \mu + \sigma\epsilon \tag{1.32}$$

where $\epsilon \sim \mathcal{N}(0, 1)$, μ, and σ are fixed vectors. Thanks to this trick, we can train our neural network through standard backpropagation. In the model of the AE (1.5.2) and VAE (1.5.3), the user has the choice of the structure of the neural networks: how many hidden layers and neurons are in each layer. Moreover, the user can choose the activation function of each neural in the network. So, many possibilities are available for the user to create his own AE or VAE.

To better understand how AE/VAE works theoretically and how to code them in PyTorch, I relied on different blog articles [24, 28].

Chapter 2

Molecules generation using graph representation

Machine learning can be used to generate new valid molecules, which can save time and money in discovering new molecules. Today, there are different machine- and deep learning generation algorithms. So, there are different ways to generate molecules, like with : VAE, GAN, normalizing flows, diffusion processes, or RNNs. These processes are based on learning the distribution of the train molecule data and are trying to generate molecules that follow the same distribution but are not in the training dataset. To learn the distribution, the deep learning algorithms are based on several characteristics: molecular shape, size, chemical composition, functional groups, or other physicochemical properties... Ultimately, the molecules generated will allow researchers to possibly discover new molecules that share some properties with molecules already known. Therefore, these algorithms are helpful in drug discovery research because they can replace old means to do it and save time and money.

To generate molecules automatically using graph representation, I relied on three articles that used different methods to do it [17, 19, 36] always based on the same generation process : diffusion process. I reuse their methods and their code, which was available on GitHub, and run their deep learning algorithms on the same dataset to compare the results.

Moreover, I relied my work on two other methods based on GAN [8, 21]. This time, I just took the results published by the author in their paper to compare them with the other article.

2.1 DiGress: Discrete denoising diffusion for graph generation

In this section, I will explain in more detail the algorithms and methods used in the DiGress article [36] to generate molecules with good metrics results.

The work in this article introduces a discrete denoising diffusion model for generating graphs with categorial node and edge attributes. Moreover, a procedure for conditioning the generation on graph-level features is also proposed in the article. Other diffusion models for graphs are proposed to embed the graphs in a continuous space. However, this idea destroys the graph's sparsity and structural information (such as connectivity or cycle counts). As a result, the continuous diffusion process can make it difficult for the denoising network to capture the structural properties of the graphs. This is why the authors of this article decided to use a discrete denoising diffusion model.

To represent graphs with categorical node and edge, the paper uses two spaces: $\mathcal{X}$ and $\mathcal{E}$, respectively, with cardinalities a and b. We note $x_i \in \mathbb{R}^a$ the one-hot encoding of node i. These encodings are organized in a matrix $X \in \mathcal{M}_{n,a}(\mathbb{R})$, where n is the number of nodes in the graph. In the same way, a tensor $E \in \mathbb{R}^{n \times n \times b}$ regroups the one-hot encoding $e_{i,j}$ of the edges of the graph. So, to represent a graph G, we note $G = (X, E)$.

Here is an overview of the process that is used to generate molecules through the DiGress algorithm:

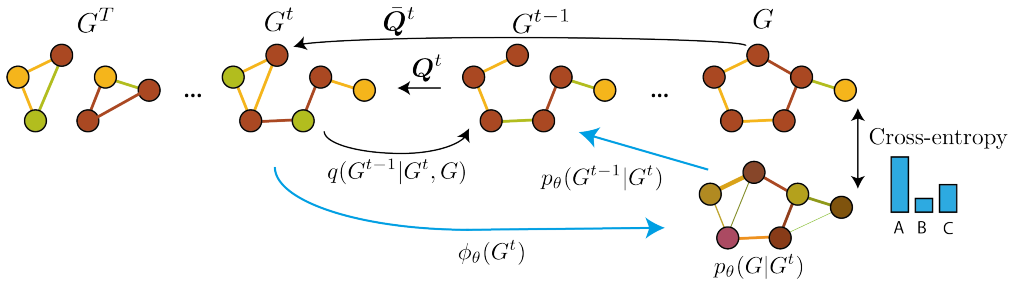


Figure 2.1: Overview of the generation process of DiGress (Image source [36])

2.1.1 Diffusion Process

Similarly to diffusion models for images, which apply noise independently to each pixel, DiGress diffuses separately on each node and edge.

So, for any node x (resp. edge e), the transition probabilities are defined by matrices:

$$[Q_X^t]_{i,j} = q(x^t = j|x^{t-1} = i) \text{ and } [Q_E^t]_{i,j} = q(e^t = j|e^{t-1} = i) \tag{2.1}$$

where q is the noise model that we add on each node and edge of the graph at each time step ($q(x^t = j|x^{t-1} = i)$ represents the probability that the node x goes from the state i to the state j by adding the noise between time steps $t-1$ and t.

The graph at the time step t is written as $G^t = (X^t, E^t)$. So, adding noise to G^{t-1} to obtain G^t can be done like that : $q(G^t|G^{t-1}) = (X^{t-1}Q_X^t, E^{t-1}Q_E^t)$ and therefore we obtain (Markovian process) :

$$q(G^t|G) = (X\overline{Q}_X^t, E\overline{Q}_E^t) \tag{2.2}$$

where $\overline{Q}_X^t = Q_X^1 \cdots Q_X^t$, $\overline{Q}_E^t = Q_E^1 \cdots Q_E^t$ and $G = (X, E)$.

The choice of the noising process is explained in more detail in the article [36] (section 4.1). They propose to use :

$$Q_X^t = \alpha^t I + \beta^t \mathbb{1}_a m_X \text{ and } Q_E^t = \alpha^t I + \beta^t \mathbb{1}_b m_E \tag{2.3}$$

This choice should result in the molecules generated being close to the train data distribution. Moreover, experimentally, these transition probabilities improve over uniform transitions.

Then, because $(\mathbb{1}m)^2 = \mathbb{1}m$, we obtain :

$$\overline{Q}^t = \overline{\alpha}^t I + \overline{\beta}^t \mathbb{1} m_X \tag{2.4}$$

where $\overline{\alpha}^t = \cos(0.5\pi(t/T + s)/(1 + s))^2$, with s a small parameter and $\overline{\beta}^t = 1 - \overline{\alpha}^t$.

2.1.2 Denoising Process

The denoising process is done by a neural network: ϕ_θ parametrized by θ. It takes a noisy graph $G^t = (X^t, E^t)$ as input and tries to predict the clean original graph $G = (X, E)$.

Let $z = f(G^t, t)$ be a vector, where f is a function that captures features of the graph at time step t (see the article for further detail). Then, $\phi_\theta(G^t, z) = \hat{p}^G = (\hat{p}^X, \hat{p}^E)$ where $\hat{p}_i^X$ is an estimation of the distribution $p_\theta(x_i|G^t)$.

Training phase

To train the ϕ_θ neural network, we optimize the cross-entropy loss l between the predicted probabilities $\hat{p}^G = (\hat{p}^X, \hat{p}^E)$ for each node and edge and the true graph G :

$$l(\hat{p}^G, G) = \sum_{1 \leq i \leq n} \text{cross-entropy}(x_i, \hat{p}_i^X) + \lambda \sum_{1 \leq i,j \leq n} \text{cross-entropy}(e_{i,j}, \hat{p}_{i,j}^X) \quad (2.5)$$

where $\lambda \in \mathbb{R}^+$ controls the relative importance of nodes and edges, and cross-entropy$(p, q) = \mathbb{E}_p(-p \log q)$ for two distributions p and q.

Algorithm 3 Training DiGress

1: A graph $G = (X, E)$
2: Sample $t \sim \mathcal{U}(1, ..., T)$
3: $G^t \sim X\bar{Q}_X^t \times E\bar{Q}_E^t$ {Sample a noisy graph}
4: $z \leftarrow f(G^t, t)$ {Structural and spectral features through f}
5: $\hat{p}^X, \hat{p}^E \leftarrow \phi_\theta(G^t, z)$
6: $optimizer.step(l_{CE}(\hat{p}^X, X) + \lambda\ l_{CE}(\hat{p}^E, E))$

Generation Phase

Once the network ϕ_θ is trained, we can use it, to estimate the reverse diffusion process : $p_\theta(G^{t-1}|G^t)$ using the network prediction $\hat{p}^G$.

$$p(G^{t-1}|G^t) = \prod_{1 \leq i \leq n} p(x_i^{t-1}|G^t) \prod_{1 \leq i,j \leq n} p(e_{i,j}^{t-1}|G^t) \quad (2.6)$$

where we use the network prediction :

$$p(x_i^{t-1}|G^t) = \int_{\mathcal{X}} p(x_i^{t-1}|x_i, G^t) dp(x_i|G^t) = \sum_{x \in \mathcal{X}} p(x_i^{t-1}|x_i = x, G^t) \hat{p}_i^X(x) \quad (2.7)$$

where we choose, $p(x_i^{t-1}|x_i = x, G^t) = q(x_i^{t-1}|x_i = x, x_i^t)$ if $q(x_i^t|x_i = x) > 0$ and is equal to zero otherwise.

Similarly, we have :

$$p(e_{i,j}^{t-1}|G^t) = \sum_{e \in \mathcal{E}} p(e_{i,j}^{t-1}|e_{i,j} = e, e_{i,j}^t) \hat{p}_{i,j}^E(x) \quad (2.8)$$

Algorithm 4 Sampling from DiGress

Sample n from the training data distribution
Sample $G^T \sim q_X(n) \times q_E(n)$

for t = T to 1 **do**
  $z \leftarrow f(G^t, t)$ {Structural and spectral features through f}
  $\hat{p}^X, \hat{p}^E \leftarrow \phi_\theta(G^t, z)$
  $p_\theta(x_i^{t-1}|G^t) \leftarrow \sum_x q(x_i^{t-1}|x_i = x, x_i^t)\ \hat{p}_i^X(x) \quad i \in 1, \dots, n$
  $p_\theta(e_{ij}^{t-1}|G^t) \leftarrow \sum_e q(e_{ij}^{t-1}|e_{ij} = e, e_{ij}^t)\ \hat{p}_{ij}^E(e) \quad i, j \in 1, \dots, n$
  $G^{t-1} \sim \prod_i p_\theta(x_i^{t-1}|G^t) \prod_{ij} p_\theta(e_{ij}^{t-1}|G^t)$
end for
return G^0

2.1.3 Conditional Generation

To force the generation of the molecules in a certain direction, the authors of the article used a regressor g_η that is trained to predict target properties y_G of a clean graph G from a noisy version of G: $g_\eta(G^t) = \hat{y}$. This regression model is trained with the following loss :

$$\min_\eta \|g_\eta(G^t) - y_G\| \tag{2.9}$$

See the original article [36] for further detail.

2.1.4 Equivariance properties

It is important to notice that $n!$ matrices can represent the same graph (ordering the nodes in another way). Therefore, to efficiently learn from this data, it is crucial to have an algorithm that takes this phenomenon into account. Hopefully, DiGress is permutation equivariant, the loss function 2.5 is permutation invariant, and DiGress is exchangeable (all permutations of generated graphs are equally likely). The proof of these claims are available in the original article for more details.

2.2 GruM : Graph Generation with Diffusion Mixture

In this section, I will explain in more detail the algorithms and methods used in the GruM article [19] to generate molecules with good metrics results.

The work in this article introduces a generative process as a mixture of endpoint-conditioned diffusion processes that is driven toward the predicted graph. Other diffusion models for graphs are learning to denoise the noisy samples. However, this

method does not explicitly learn the graph structures to be generated. To tackle this limitation, the authors of this article proposed a generative framework that models the topology of graphs by explicitly learning the final graph structures of the diffusion process. The goal of the diffusion process in this article is to directly learn the final graph and its structure instead of learning how to denoise a noisy version of the original graphs. However, predicting the final graph structure of the diffusion process is difficult since the prediction would be highly inaccurate in the early steps of the process, and such an inaccurate prediction may lead the process in the wrong direction, resulting in invalid results.

To resume, they propose a novel framework that explicitly models the graph topology, by learning the prediction of the resulting graph of the generative process which is represented as a weighted mean of graph data.

To represent graphs with a categorical node and edge, the paper uses two spaces. We note the node matrix $X \in \mathcal{M}_{N,F}(\mathbb{R})$ where N is the number of nodes in the graph and F is the dimension of the node's features. In the same way, the edge matrix $A \in \mathcal{M}_{N,N}(\{0,1,2,3\})$ represents the adjacency matrix of the graph (this defines the connection type between nodes): 0 for no bond, 1 for the single bond, 2 for the double bond, and 3 for the triple bond. So, to represent a graph G, we note $G = (X, A)$.

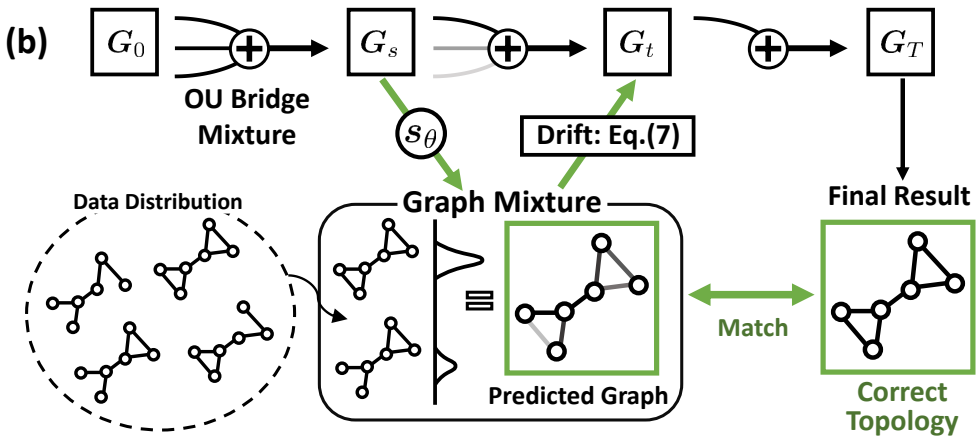


Figure 2.2: Overview of the generation process of GruM (Image source : [19])

2.2.1 Diffusion Process

The goal of this article is to directly predict the final graph of the diffusion process that transports a prior distribution to the data distribution Π^*. For a graph diffusion process represented as $G_t = (X_t, A_t)_{t\in[0,T]}$, we aim to predict G_T in Π^*

given the current state G_t. However, identifying the exact G_T at the early stage of the process is problematic since the prediction based on G_t of almost no information would be highly inaccurate, and could lead the process in the wrong direction.

To address this problem, a different approach is used to predict the probable graph, which is defined as a weighted mean of all the possible final results. Since the probability of a graph g being the final result is equal to the transition probability of the process denoted as $p_{T|t}(g|\cdot)$, we define the probable graph given the current state G_t via the expectation of the graphs as follows:

$$D(G_t, t) = \int g \cdot p_{T|t}(g|G_t)dg \tag{2.10}$$

which we refer to as the graph mixture of the process.

2.2.2 Denoising Process

Our goal is to design a generative model that explicitly learns the graph topology. To this end, we leverage the Ornstein Uhlenbeck (OU) bridge mixture parameterized by the graph mixture, where we estimate the graph mixture using a neural network $s_\theta(\cdot, t)$ that corresponds to directly learning the graph structures.

Training phase

The simplified loss function of the neural network $s_\theta(\cdot, t)$ corresponds to :

$$\mathcal{L}(\theta) = \mathbb{E}_{G\sim\mathbb{Q}^\Pi}\left(\frac{1}{2}\int_0^T c^2\|s_\theta(G_t, t) - G_T\|^2 dt\right) \tag{2.11}$$

where c is a coefficient function.

Algorithm 5 Training of GruM

Input: Model s_θ, constant ϵ

For each epoch:

1: Sample graph G from the training set
2: $N \leftarrow$ number of nodes of G
3: Sample $t \sim [0, T - \epsilon]$ and $G_0 \sim \mathcal{N}(0, \mathbf{I}_N)$
4: Sample $G_t \sim p_{t|0,T}(G_t|G_0, G)$ {Section A.6 from the original article}
5: $\mathcal{L}_\theta \leftarrow c^2\|s_\theta(G_t, t) - G\|^2$
6: Update θ using $\mathcal{L}_\theta$

Generation Phase

Using the trained model s_θ to compute the drift η_θ of the parameterized mixture process. Note that we generate the node features and the adjacency matrices simultaneously using the system of SDEs.

Algorithm 6 Sampling of GruM

Input: Trained model s_θ, number of sampling steps K, diffusion step size $\mathrm{d}t$

1: Sample number of nodes N from the training set.
2: $G_0 \sim \mathcal{N}(0, I_N)$ {Start from noise}
3: $t \leftarrow 0$
4: **for** $k = 1$ to K **do**
5: $\quad \eta \leftarrow \alpha\sigma_t^2 G_t + \frac{\sigma_t^2}{v_t}\left(\frac{1}{u_t}s_\theta(G_t, t) - G_t\right)$
6: $\quad w \sim \mathcal{N}(0, I_N)$
7: $\quad G_{t+\mathrm{d}t} \leftarrow \eta\mathrm{d}t + \sigma_t\sqrt{\mathrm{d}t}w$ {Euler-Maruyama see Section 2.9.3}
8: $\quad t \leftarrow t + \mathrm{d}t$
9: **end for**
10: $G \leftarrow \texttt{quantize}(G_t)$ {Transform the real tensors (X,A) in categorical tensors to corresponds to a molecule}
11: **return** Graph G

2.3 CDGS : Conditional Diffusion Based on Discrete Graph Structures for Molecular Graph Generation

In this section, I will explain in more detail the algorithms and methods used in the CDGS article [17] to generate molecules with good metrics results.

The method of this article, is to noise the original graph. This process is denoted by a Stochastic Differential Equation (SDE). The reverse time SDE is theoretically known. However a term of this equation can not be computed in practice. Therefore, this part is estimated by a neural network. Then, the graph $G = (X, A)$ can be computed by the reverse time process after discretization.

To represent graphs with categorical node and edge, the paper uses two tensors: $X \in \mathcal{M}_{N,F}(\mathbb{R})$ which represent the node features of dimensions F and $A \in \mathbb{R}^{N\times N\times 2}$ which represents the edge information: there is a channel used for the edge existence and another for type of edge. We note $\bar{A} \in \{0, 1\}^{N\times N}$ the matrix channel used for the existence of edges.

2.3.1 Diffusion process

Let be $x_0 \in \mathbb{R}^d$ and a forward process $\{x_t\}_{t\in[0,T]}$, we note

$$q_{0t}(x_t|x_0) = \mathcal{N}(x_t|\alpha_t x_0, \sigma_t^2 I) \ , \tag{2.12}$$

where $\alpha_t, \sigma_t \in \mathbb{R}^+$ are time-dependant differentiable functions. α_t and σ_t are usually chosen to ensure that $q_T(x_T) \approx \mathcal{N}(0, I)$. By learning to reverse such a process, the diffusion model generates new samples from the prior distribution.

The diffusion process of this article follow the following stochastic differential equation for $t \in [0, T]$:

$$\mathrm{d}G_t = f(t)G_t\mathrm{d}t + g(t)\mathrm{d}w_t \tag{2.13}$$

where $f(t) = \frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t}$ is the drift coefficient, $g^2(t) = \frac{\mathrm{d}\sigma_t^2}{\mathrm{d}t} - 2\frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t}\sigma_t^2$ is the diffusion coefficient, and w_t is a standard Wiener process.

2.3.2 Denoising process

The reverse-time SDE from time T to 0 corresponding to equation 2.13 is denoted as thanks to Anderson's reverse-time SDE theorem [9]:

$$\mathrm{d}G_t = [f(t)G_t - g^2(t)\nabla_G \log q_t(G_t)]\mathrm{d}_t + g(t)\mathrm{d}\bar{w}_t \tag{2.14}$$

where $\bar{w}_t$ is the reverse-time standard Wiener process.

We further split the reverse-time SDE into two parts that share the drift and diffusion coefficients as follows:

$$\begin{aligned} \mathrm{d}X_t &= [f(t)X_t - g^2(t)\nabla_X \log q_t(X_t, A_t)]\mathrm{d}_t + g(t)\mathrm{d}\bar{w}_t^1 \\ \mathrm{d}A_t &= [f(t)A_t - g^2(t)\nabla_A \log q_t(X_t, A_t)]\mathrm{d}_t + g(t)\mathrm{d}\bar{w}_t^2 \end{aligned} \tag{2.15}$$

Training phase

We use a neural network $\epsilon_\theta(G_t, \bar{A}_t, t)$ where the node output of the neural network is denoted by $\epsilon_{\theta,X}(G_t, \bar{A}_t, t)$ to estimate $-\sigma_t\nabla_X \log q_t(X_t, A_t)$, and the edge output is denoted by $\epsilon_{\theta,A}(G_t, \bar{A}_t, t)$ to estimate $-\sigma_t\nabla_A \log q_t(X_t, A_t)$.

The model is optimized by the objective as follows:

$$\min_\theta \mathbb{E}_t \left(w(t)\mathbb{E}_{G_0}\mathbb{E}_{G_t|G_0} \left(\|\epsilon_{\theta,X}(G_t, \bar{A}_t, t) - \epsilon_X\|_2^2 + \|\epsilon_{\theta,A}(G_t, \bar{A}_t, t) - \epsilon_A\|_2^2 \right) \right) \tag{2.16}$$

where $w(t)$ is a given positive weighting function, ϵ_X and ϵ_A are the sampled Gaussian noise, and $G_t = (\alpha_t X_0 + \sigma_t\epsilon_X, \alpha_t A_0 + \sigma_t\epsilon_A)$.

The neural network $\epsilon_\theta(G_t, \bar{A}_t, t)$ is explained in more detail in the original article [17]. To explain it briefly, this network uses an attention layer, and the goal of the noise prediction model is to extract the global context and the local node-edge dependency from intermediate graph states.

Algorithm 7 Optimizing CDGS

Require: original graph data $G_0 = (X_0, A_0)$, graph noise prediction model ϵ_θ, schedule function $\alpha(\cdot)$ and $\sigma(\cdot)$, quantized function $quantize(\cdot)$

1: Sample $t \sim \mathcal{U}(0,1], \epsilon_X \sim \mathcal{N}(0,I), \epsilon_A \sim \mathcal{N}(0,I)$
2: $G_t = (X_t, A_t) \leftarrow (\alpha(t)X_0 + \sigma(t)\epsilon_X, \alpha(t)A_0 + \sigma(t)\epsilon_A)$
3: $\bar{A}_t \leftarrow quantize(A_t)$
4: $\epsilon_\theta^X, \epsilon_\theta^A \leftarrow \epsilon_\theta(G_t, \bar{A}_t, t)$
5: $\min_\theta \|\epsilon_\theta^X - \epsilon_X\|_2^2 + \|\epsilon_\theta^A - \epsilon_A\|_2^2$

Generation phase

Once the neural network ϵ_θ is trained, we can easily estimate $-\sigma_t \nabla_X \log q_t(X_t, A_t)$ and $-\sigma_t \nabla_A \log q_t(X_t, A_t)$.

Using Anderson's reverse-time SDE theorem [9], to simulate the reverse time SDE (equation 2.15), we separate the interval $[0, T]$ into an equal sub-interval of size Δt and use the Euler-Maruyama method:

$$\begin{aligned} X_{t-1} &\leftarrow X_t - (f(t)X_t + \tfrac{g(t)^2}{\sigma(t)}\epsilon_\theta^X)\Delta t + g(t)\sqrt{\Delta t}\epsilon_X \\ A_{t-1} &\leftarrow A_t - (f(t)A_t + \tfrac{g(t)^2}{\sigma(t)}\epsilon_\theta^A)\Delta t + g(t)\sqrt{\Delta t}\epsilon_A \end{aligned} \tag{2.17}$$

To have more details about the Euler-Maruyama method, see section 2.9.3, it's a numerical method to approach the solution of a SDE. Here is an overview of steps to generate new molecules through the CDGS method.

Algorithm 8 Sampling from CDGS with the Euler-Maruyama method

Require: number of time steps N, graph noise prediction model ϵ_θ, drift coefficient function $f(\cdot)$, diffusion coefficient function $g(\cdot)$, schedule function $\sigma(\cdot)$, quantized function $quantize(\cdot)$, post-processing function $post(\cdot)$

1: Sample initial graph $G \leftarrow (X \sim \mathcal{N}(0,I), A \sim \mathcal{N}(0,I))$,
2: $\Delta t = \frac{T}{N}$
3: **for** $i \leftarrow N$ to 1 **do**
4: $\quad \bar{A} \leftarrow quantize(A)$
5: $\quad \epsilon_X \sim \mathcal{N}(0,I), \epsilon_A \sim \mathcal{N}(0,I)$
6: $\quad t \leftarrow i\Delta t$
7: $\quad \epsilon_\theta^X, \epsilon_\theta^A \leftarrow \epsilon_\theta(G, \bar{A}, t)$
8: $\quad X \leftarrow X - (f(t)X + \frac{g(t)^2}{\sigma(t)}\epsilon_\theta^X)\Delta t + g(t)\sqrt{\Delta t}\epsilon_X$
9: $\quad A \leftarrow A - (f(t)A + \frac{g(t)^2}{\sigma(t)}\epsilon_\theta^A)\Delta t + g(t)\sqrt{\Delta t}\epsilon_A$
10: **end for**
11: **return** $post(X, A)$

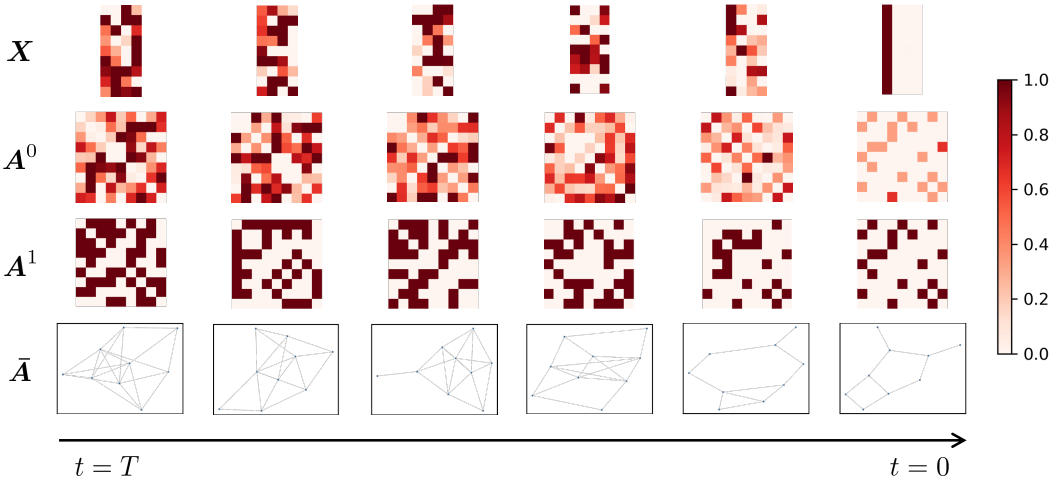


Figure 2.3: Visualization at different steps in the reverse generative process from a model trained on QM9. X is the node feature matrix, A^0 is the edge type matrix, and A^1 is the quantized edge existence matrix. (Image source [17])

2.4 MolGAN: An implicit generative model for small molecular graphs

In this section, I will explain the idea behind the MolGAN method [8].

The model named MolGAN is explained on the figure 2.4:

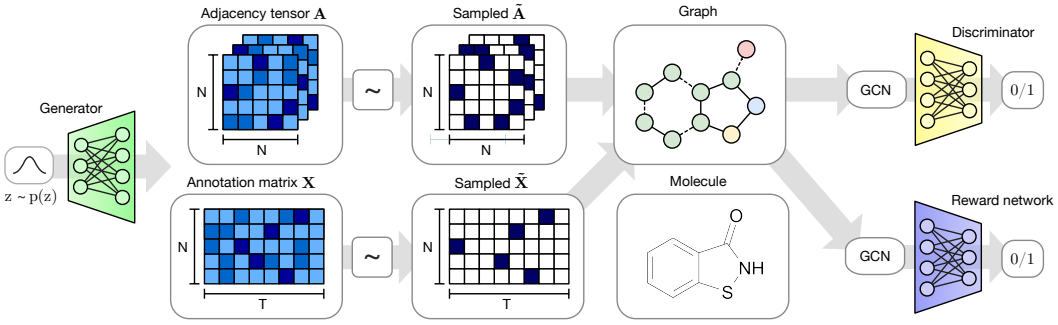


Figure 2.4: MolGAN model (Image source : [8])

First, like a classical GAN (see section 2.7.1), a sample is taken from a prior distribution, and then the generator gives two matrices: $X \in \mathbb{R}^{N \times T}$ that defines atom types, and $A \in \mathbb{R}^{N \times N \times Y}$ that defines bond types (where N is the number of nodes of the graph generated, T is the number of atom types, and Y is the number of bond types).

Then, we transform the tensors A and X to categorical tensors $\tilde{A}$ and $\tilde{X}$ (we put a one on the cell that has the most probability to be chosen according to A for each line for $\tilde{X}$ and for each third dimension for $\tilde{A}$). This is exactly a representation of a molecule through a graph, as explained in the introduction of the report.

Then, the graph is processed by the discriminator to guess if it is a real sample or a sample generated by the generator. Moreover, the graph is processed by the reward network, which forces the generation of graphs in a certain direction: at most valid, unique, and novel molecules. This reward network is part of reinforcement learning.

Here is more detail about the Generator, the Discriminator and the Reward Network. The Generator is a simple multi-layer perceptron (MLP), even if other articles use RNN (see Section 2.7.4). To see the exact structure of the MLP, see the original article in the section Generator Architecture [8]. Both Discriminator and Reward network take a graph as input and a scalar as output. They both have the same structure but don't share the same parameters. They are composed of convolutional and traditional layers, for more detail, see the article [8].

Training phase

The discriminator is trained using the GAN objective, while the generator uses a linear combination of the GAN loss and the RL loss :

$$L(\theta) = \lambda \cdot L_{WGAN}(\theta) + (1 - \lambda) \cdot L_{RL}(\theta) \tag{2.18}$$

where $\lambda \in [0, 1]$ is a hyperparameter that regulates the weight of each component.

To obtain their results, the authors used the best λ parameter : determined via the model with the maximum sum of valid, unique, novel, and solubility scores.

2.5 Pure Transformer Encoders Make an Efficient Discrete GAN for De Novo Molecular Generation

In this section, I will explain the idea behind the article Pure Transformer Encoders Make an Efficient Discrete GAN for De Novo Molecular Generation method called TenGAN [21].

The model named TenGAN is explained on the figure 2.5.

In this method, the numerical representation of molecules for generation is the SMILES representation, which is processed as text with the famous attention

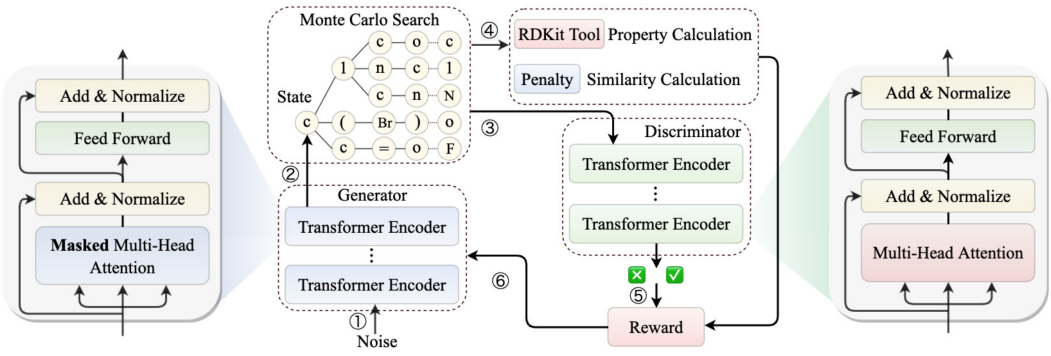


Figure 2.5: Architecture of TenGAN model (Image source [21])

phenomenon that is used in transformers, which is used in the language models like ChatGPT.

So, like a classical GAN (see section 2.7.1), a sample is taken from a prior distribution. Then, the generator takes this sample as input and produces SMILES substrings. Then, the generated substrings are complemented using the Monte Carlo (MC) search. The complete SMILES strings are then used as input by the discriminator, which has to predict the probability that the SMILES string belongs to the real and not to the generated dataset. At the same time, the SMILES string is penalized according to the repetition rate to avoid producing the same molecule, and the RDKit library is used to calculate the value of their chemical properties. Finally, the probabilities, penalties, and property scores are jointly used as rewards to update the parameters of the generator with RL.

2.6 Comparison of the results of the five articles

Here is a table that contains information about the dataset named QM9 that I used to compare the methods of the 5 articles [36, 19, 17, 8, 21] to generate molecules.

Dataset	Number of molecules	Number of nodes	Number of node types	Number of edge types
QM9	133,885	$1 \leq \lvert V \rvert \leq 9$	4	3

Table 2.1: Dataset information

If I had more time, I would do it on several datasets, like the ZINC250K dataset, which contains molecules with more node types.

For the article that follow a diffusion process, I used the code of the authors of each article and ran them on the same dataset. For DiGress, I trained the model on the QM9 dataset, while for GruM and CDGS, I used checkpoints given by the

author (which were trained on the same QM9 dataset). Then I generated 10000 molecules for each article, and I got these results:

Article	Validity (%) ↑	Uniqueness (%) ↑	Novelty (%) ↑	FCD ↓
DiGress	99.12	96.46	34.14	-
CDGS	99.71	96.98	69.54	0.213
GruM	99.42	97.20	25.24	0.109

Table 2.2: Results for each article trained on the QM9 Dataset by myself (Diffusion Process)

Article	Validity (%) ↑	Uniqueness (%) ↑	Novelty (%) ↑	FCD ↓
DiGress	99.0	96.2	33.4	-
CDGS	99.68	96.83	69.62	0.200
GruM	99.69	-	-	0.108

Table 2.3: Results for each article given by the paper

However, we should be careful with the novelty metrics for the QM9 dataset. Indeed, the QM9 dataset is only an enumeration of small molecules that satisfy a given set of constraints. So, generating molecules outside this set is not necessarily a good sign that the network has correctly captured the data distribution.

To compare the results obtained by myself and those given in the article, we can see that the results are very close to each other. The little difference can be explained by the randomness phenomenon that is the source of the generation of the molecules.

Article	Validity (%) ↑	Uniqueness (%) ↑	Novelty (%) ↑	FCD ↓
MolGAN	98	2.3	-	-
TenGAN	97.33	52.16	97.7	

Table 2.4: Results for each article trained on the QM9 Dataset given by the paper (methods based on GAN)

We can see that on the QM9 dataset, the generative methods based on GAN have more difficulties generating a dataset with a lot of unique molecules. However, in the dataset of unique molecules, a huge amount is new and wasn't in the training dataset. These are some of the advantages and drawbacks of each method.

2.7 Generation algorithms

In this section, I will explain different deep learning algorithms that can be used to generate molecules. In each case, reinforcement learning can be used in addition to the original algorithm to generate valid molecules. In general, we use conditional learning to help the original algorithm generate molecules that verify some properties.

2.7.1 GAN

Generative Adversarial Networks (GAN) are possible algorithms that can generate samples like images or molecules, for example. I will explain in more detail how a GAN really works. To understand how GAN works, I relied on several articles and websites [18, 32, 14].

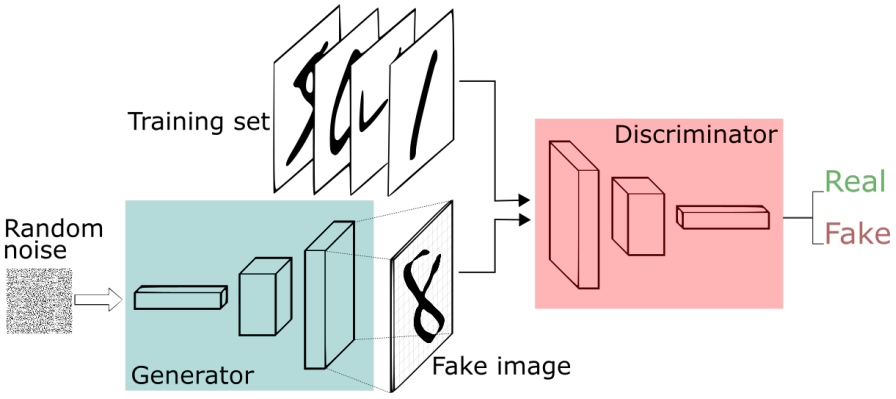


Figure 2.6: Structure of a GAN (Image source [32])

A GAN is composed of two neural networks: a generator (G) and a discriminator (D). In the case of images, the generator and the discriminator are constituted by convolutional layers. The goal of the generator is to generate a fake image from a random noise that follows most of the distribution of the train images and that will mislead the discriminator in it's objective. The goal of the discriminator is to be able to differentiate a real image from a fake image.

First, we train the two neural networks (G and D) together by minimizing the following loss function:

$$\min_{\theta} \max_{\phi} \mathbb{E}_{x \sim p_{data}(x)}[\log(D_\phi(x))] + \mathbb{E}_{z \sim p_z(z)}[1 - \log(D_\phi(G_\theta(z)))] \tag{2.19}$$

Then, to generate new samples, we just have to apply the generator network to random noise, and it will generate a new fake sample that normally follows the

train dataset distribution. Therefore, the generated sample looks like the data from the training dataset.

Unfortunately, a common drawback of the GAN is that it has difficulties to converge and to find the optimal value of (θ, ϕ). This problem can conduct to bad generation.

There exists some variant of the original GAN, which is explained in this article [14]. The variants are often adapted to a specific generational task. For example, to generate molecules, the MolGAN [8] is a variant of a GAN that uses reinforcement learning to force the generator to generate valid molecules. More explanation about this method in the section 2.4.

2.7.2 VAE

Like explained in section 1.5.3, VAE can be used for their latent space to represent data in a shorter dimension space. However, they can be used to generate new data too. Indeed, once the VAE is trained with the train dataset, we know that the data follows the distribution $\mathcal{N}(\mu, \sigma)$ in the latent space. Therefore, we just have to compute a new sample z like that : $z \sim \mathcal{N}(\mu, \sigma)$ in the latent space. Then, to obtain the new sample in the input space, we just have to apply the decoder on the sample generated in the latent space.

2.7.3 Diffusion processes

A diffusion process is a process that is done in two steps. The first step is the noising process. In this part, we take a sample from our train dataset and add noise to it a certain number of times. For a large enough number of steps, the sample distribution follows a known distribution (an isotropic Gaussian, for example). Then, there is the denoising process, the goal is to predict or estimate through a neural network, for example, the noise added at each step. In the final step, we are training our neural network to obtain the original sample at the original time step. Then, to generate a new sample, we start with an isotropic Gaussian and apply the trained neural network to it to obtain a new sample that follows the same distribution as the train samples.

To better understand the process here is a scheme of the process :

So, to explain with the figure 2.7, at the beginning we have a training sample which corresponds to the portrait of the man here (x_0). At each time step t, we had some noise to the original data (written $q(x_t|x_{t-1})$) and for a big time step T, we should obtain a known distribution. During the training, our goal is to estimate the noise added at each step to be able to reverse the process : we want to estimate

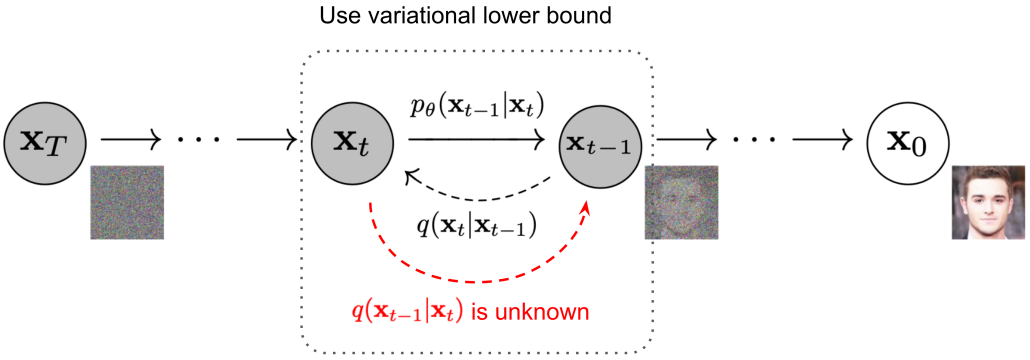


Figure 2.7: Structure of a diffusion process (Image source : [38])

$p_\theta(x_{t-1}|x_t)$ through mathematical calculus or a neural network. Then, to generate a new sample, we start from x_T which follow a known distribution, and we add the reverse noise estimation to obtain a new sample .

2.7.4 RNN

Recurrent neural networks (RNN) are a particular type of neural network that captures information about order in the sequence, tries to capture long term dependencies, and is able to process sequences of different lengths.

RNN are auto-regressive processes in contrast to diffusion processes, which are one-shot generation models. Indeed, RNNs are a particular type of neural network that is made to process sequences of data. The SMILES representation of molecules can be seen as a sequence of characters. Therefore, RNNs can be a means to process the molecules and generate new ones.

Here is a scheme of the structure of a RNN :

We can see that, the structure is identical to that of a feed forward neural network. Moreover, we can see that the neurons in the same layer are connected from the start to the end : this is the recurrent part of the network. However, RNN suffers from different problems like exploding or vanishing gradients, which cause problems with dependencies in the sequence. To solve this problem, we can use some gates like GRU, or LSTM, which are helpful.

Even so, RNNs are slow because they are not able to do parallelization. This is why the attention mechanism and the transformers became so famous in the last few years [35].

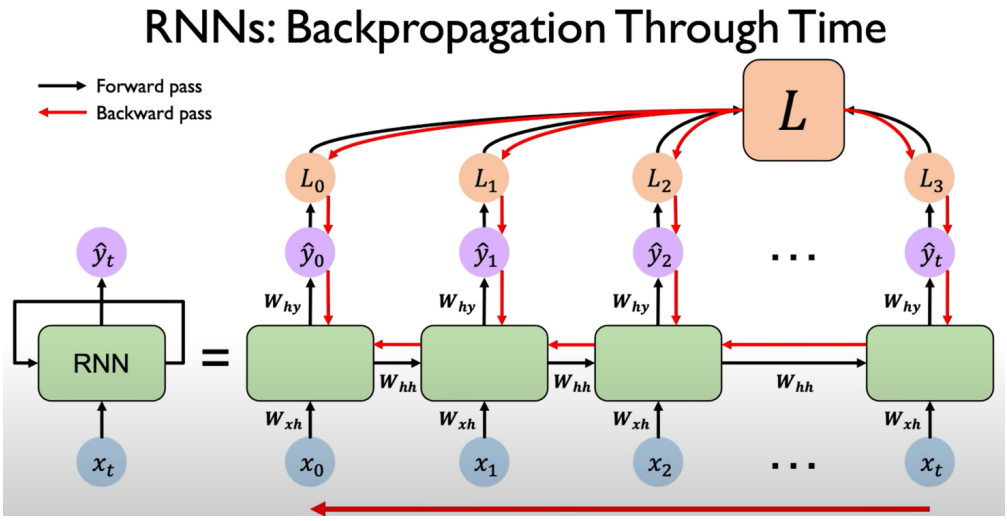


Figure 2.8: Structure of a RNN (Image source : [20])

2.8 Molecules generation metrics

In this part, I will explain in more detail the useful metrics that can be used in the case of molecule generation.

2.8.1 Validity

The validity represents the number of valid molecules generated over the total number of molecules generated. This metric takes values between 0 and 1. The score is better when the validity is closer to 1, which corresponds to a high number of valid molecules generated by the algorithm.

If we note our dataset D generated by our algorithm, the validity corresponds to:

$$\text{Validity} = \frac{\#\text{of valid molecules in D}}{\#\text{of molecules in D}} \tag{2.20}$$

To test if a generated molecule is valid or not, we check that no atoms in the molecule have exceeded their possible valency.

2.8.2 Uniqueness

The uniqueness metric corresponds to the proportion of unique molecules generated in our dataset by the algorithm. If we note our dataset D generated by our

algorithm and D_V the dataset of valid molecules generated by our algorithm, the uniqueness corresponds to:

$$\text{Uniqueness} = \frac{\#\text{of unique molecules in } D_V}{\#\text{of molecules in } D_V} \tag{2.21}$$

Uniqueness takes values between 0 and 1. The score is better when the uniqueness is closer to 1, which corresponds to a high number of unique molecules generated by the algorithm in the generated dataset.

2.8.3 Novelty

The novelty metrics correspond to the proportion of new molecules created by the generation algorithm that were not in the training dataset.

Indeed, if we note T the training dataset, and $D_{V\&U}$ the dataset of valid and unique molecules generated, the novelty corresponds to:

$$\text{Novelty} = \frac{\#\text{of molecules in } D_{V\&U} \text{ and not in T}}{\#\text{of molecules in } D_{V\&U}} \tag{2.22}$$

The novelty metric takes values between 0 and 1. The score is better when the uniqueness is closer to 1, which corresponds to a high number of novel molecules generated by the algorithm in the generated dataset.

2.8.4 FCD

The Fréchet ChemNet Distance (FCD) [26] is a metric that represents the distance between the real distribution $p_w(\cdot)$ and the distribution $p(\cdot)$ of molecules from a generative model. To obtain a numerical representation of each molecule, we use the activation of the last layer of "ChemNet" : a neural network. Then, we calculate the mean (m) and the covariance (C) of the activation of these two distributions. Assuming that these two distributions follow a Gaussian distribution : $\mathcal{N}(m, C)$. The FCD metric can be computed like this :

$$d^2((m, C), (m_w, C_w)) = \|m - m_w\|_2^2 + Tr(C + C_w - 2(CC_w)^{1/2}) \tag{2.23}$$

where Tr is the trace of a matrix.

This metric is used to measure the similarity between the distribution of the training set (the real distribution) and the generated set. Then, the FCD score is better when it is lower.

As said before, the FCD utilizes the penultimate layer of a deep neural network called "ChemNet", which was trained to predict drug activities. Thus, the FCD

metric takes into account chemically and biologically relevant information about molecules, and also measures the diversity of the set via the distribution of generated molecules.

2.9 Mathematical definitions

Here is a section to do some reminders on probabilities (random variables) and stochastic differential equations (SDE).

2.9.1 Wiener Process

The Wiener process is described as follows :

1. $W_0 = 0$ almost surely.
2. Let be $u \geq 0$, for every $t > 0$, $W_{t+u} - W_t$ is independent from the past values W_s $(s < t)$.
3. $W_{t+u} - W_t \sim \mathcal{N}(0, u)$.
4. W_t is almost surely continuous in t.

2.9.2 Ornstein–Uhlenbeck process

The Ornstein–Uhlenbeck process X_t is defined by the following stochastic differential equation:

$$dX_t = \theta(\mu - X_t)dt + \sigma dW_t \tag{2.24}$$

where $\theta > 0$ and $\sigma > 0$ are parameters, W_t is the Wiener process and μ is a constant.

2.9.3 Euler-Maruyama method

The Euler-Maruyama (EM) method is a method to numerically approach the solution of a SDE, like the ordinary Euler method for an ODE.

Let be the following SDE :

$$\mathrm{d}X_t = a(X_t, t)\,\mathrm{d}t + b(X_t, t)\,\mathrm{d}W_t, \tag{2.25}$$

where W_t is the standard Wiener process and $X_0 = x_0$ and a, b are two functions.

If we want to numerically approach the solution of this SDE with the EM method on the interval $[0, T]$, we can do it with the following Markov chain Y. First, we separate the interval $[0, T]$ into N equal sub-interval of size Δt ($\tau_i = i * \Delta t$). Then, we set $Y_0 = x_0$ and for $n \in 1, \ldots, N-1$, we define:

$$Y_{n+1} = Y_n + a(Y_n, \tau_n)\,\Delta t + b(Y_n, \tau_n)\,\Delta W_n, \tag{2.26}$$

where $\Delta W_n = W_{\tau_{n+1}} - W_{\tau_n} \sim \mathcal{N}(0, \Delta t)$

Conclusion

To conclude, this project has been a means for me to explore and learn a lot of different things. I discovered a lot of things, and it was very rewarding.

During this project, I had the time to understand deep learning generation and clustering algorithms and general machine learning networks like RNNs or transformers. I learned new mathematical tools like the Wiener process and the resolution of SDE. I understood the importance of the choice of metrics for evaluating the methods used. Finally, I learned how to transform the mathematical knowledge into practice with Python and some libraries like scikit-learn, PyTorch, and TensorFlow. In the first part (clustering), I relied on the code of the authors and modified it to obtain my own results. In the second part, I just ran and understood the code of the authors. In all cases, it has been a means for me to better understand the method explained in the article.

To conclude the work done in this project, I proposed in this report several deep learning methods for molecular clustering and generation and explained a little bit how these methods work with the math behind them. Moreover, I explained the metrics used to compare the results and the results that I obtained myself using their methods with Python.

The clustering and generation of molecular fields are very large and have not yet been totally explored. This is why there is still a lot to do. For example, another step for generation algorithms will be to take in input several forms of the same molecules: a SMILES representation, a graph representation, or an image, and from these multi-modalities, generate new molecules. The multi-modality can help the model better understand the structure and the bonds of the molecule because of the different points of view of the modalities, and the multi-modal model can benefit from each of their advantages.

Bibliography

[1] Alexander Amini. *MIT Introduction to Deep Learning | 6.S191*. Apr. 29, 2024. URL: https://www.youtube.com/watch?v=ErnWZxJovaM (visited on 04/29/2024).

[2] Nadège Arnaud. *Zotero et bibliographie LateX*. Nov. 20, 2019.

[3] T. Calinski and J. Harabasz. "A dendrite method for cluster analysis". In: *Communications in Statistics - Theory and Methods* (1974).

[4] D. Comaniciu and P. Meer. "Mean shift: a robust approach toward feature space analysis". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (May 2002).

[5] Datasans. *K-Means Cluster: Okay You Built the Model, Then What??? (No Math)*. Medium. Jan. 14, 2023. URL: https://datasans.medium.com/k-means-cluster-okay-you-built-the-model-then-what-no-math-a0e710b42252 (visited on 04/29/2024).

[6] Laurianne David et al. "Molecular representations in AI-driven drug discovery: a review and practical guide". In: *Journal of Cheminformatics* (Sept. 17, 2020). URL: https://doi.org/10.1186/s13321-020-00460-5.

[7] David Davies and Don Bouldin. "A Cluster Separation Measure". In: *Pattern Analysis and Machine Intelligence* (May 1, 1979).

[8] Nicola De Cao and Thomas Kipf. "MolGAN: An implicit generative model for small molecular graphs". In: *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models* (2018).

[9] P Dhariwal and A Nichol. *Diffusion models are SOTA*. 2024.

[10] Iakovos Evdaimon et al. *Neural Graph Generator: Feature-Conditioned Graph Generation using Latent Diffusion Models*. 2024.

[11] Kurt De Grave Fabrizio Costa. "Fast Neighborhood Subgraph Pairwise Distance Kernel". In: *Proceedings of the 27 th International Conference on Machine Learning, Haifa, Israel* (2010).

[12] Emmanuel Franck. *Auto-encoder et réduction de dimension*. URL: https://irma.math.unistra.fr/~franck/cours/SciML/output/html/chapAPsec5.html (visited on 04/29/2024).

[13] Kavya Gajjar. *Cluster Analysis with DBSCAN : Density-based spatial clustering of applications with noise*. Analytics Vidhya. Oct. 15, 2020. URL: https://medium.com/analytics-vidhya/cluster-analysis-with-dbscan-density-based-spatial-clustering-of-applications-with-noise-6ade1ec23555 (visited on 04/29/2024).

[14] Ian J. Goodfellow et al. *Generative Adversarial Networks*. June 10, 2014. URL: http://arxiv.org/abs/1406.2661.

[15] Hamid Hadipour et al. "Deep clustering of small molecules at large-scale via variational autoencoder embedding and K-means". In: *BMC Bioinformatics* (2022).

[16] Saiveth Hernández-Hernández and Pedro J. Ballester. "On the Best Way to Cluster NCI-60 Molecules". In: *Biomolecules* (2023).

[17] Han Huang et al. *Conditional Diffusion Based on Discrete Graph Structures for Molecular Graph Generation*. 2023.

[18] *Introduction | Machine Learning*. URL: https://developers.google.com/machine-learning/gan?hl=fr (visited on 05/03/2024).

[19] Jaehyeong Jo, Dongki Kim, and Sung Ju Hwang. *Graph Generation with Destination-Driven Diffusion Mixture*. 2023.

[20] Authur Lee. *MIT 6.S191: Recurrent Neural Networks*. Dec. 2, 2021. URL: https://authurwhywait.github.io/blog/2021/12/02/introduction_to_dl02/ (visited on 05/28/2024).

[21] Chen Li and Yoshihiro Yamanishi. "TenGAN: Pure Transformer Encoders Make an Efficient Discrete GAN for De Novo Molecular Generation". In: *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*. PMLR, Apr. 18, 2024.

[22] *MIT Deep Learning 6.S191*. MIT Deep Learning 6.S191. URL: http://introtodeeplearning.com (visited on 04/29/2024).

[23] *ML | Mean-Shift Clustering*. GeeksforGeeks. Section: Machine Learning. May 15, 2019. URL: https://www.geeksforgeeks.org/ml-mean-shift-clustering/ (visited on 04/29/2024).

[24] Matthew N. Bernstein. *Variational autoencoders*. Mar. 14, 2023. URL: https://mbernste.github.io/posts/vae/ (visited on 04/29/2024).

[25] Michael Phi. *Illustrated Guide to LSTM's and GRU's: A step by step explanation*. Medium. June 28, 2020. URL: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 (visited on 05/23/2024).

[26] Kristina Preuer et al. *Fréchet ChemNet Distance: A metric for generative models for molecules in drug discovery*. URL: http://arxiv.org/abs/1803.09518.

[27] *PyTorch*. PyTorch. URL: https://pytorch.org/ (visited on 04/29/2024).

[28] Clément Rambour, Nicolas Audebert, and Nicolas Thome. *Travaux pratiques : auto-encodeurs variationnels — RCP211 - Artificial Intelligence Certificate - Cnam*. URL: https://cedric.cnam.fr/vertigo/cours/RCP211/TP7-AEetVAE.html (visited on 04/29/2024).

[29] Joseph Rocca. *Understanding Variational Autoencoders (VAEs)*. Medium. Mar. 21, 2021. URL: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73 (visited on 04/29/2024).

[30] Peter J. Rousseeuw. "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis". In: *Journal of Computational and Applied Mathematics* (Nov. 1987).

[31] *scikit-learn: machine learning in Python — scikit-learn 1.4.2 documentation*. URL: https://scikit-learn.org/stable/ (visited on 04/29/2024).

[32] Thalles Santos Silva. "A Short Introduction to Generative Adversarial Networks". In: *https://sthalles.github.io* (2017). URL: https://sthalles.github.io/intro-to-gans/ (visited on 05/19/2024).

[33] Frederic Sur. *Introduction à l'apprentissage automatique*. 2023.

[34] *The RDKit Documentation — The RDKit 2024.03.1 documentation*. URL: https://www.rdkit.org/docs/index.html (visited on 04/29/2024).

[35] Ashish Vaswani et al. *Attention Is All You Need*. Aug. 1, 2023. URL: http://arxiv.org/abs/1706.03762.

[36] Clement Vignac et al. "DiGress: Discrete Denoising diffusion for graph generation". In: *The Eleventh International Conference on Learning Representations*. 2023.

[37] David Weininger. "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules". In: *Journal of Chemical Information and Computer Sciences* (Feb. 1, 1988).

[38] Lilian Weng. *What are Diffusion Models?* July 11, 2021. URL: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/ (visited on 04/29/2024).