MODAL INF473N : ALGORITHMIQUE POUR LES NANOSCIENCES

Machine Learning : Modèles de prédiction des caractéristiques des matériaux

31 mai 2022

Encadrant : Stephane Redon
Elèves : Loïc Thomas, Hamza Tarmadi, Naoufal Khaloui, Mohamed Lakhnichi

TABLE DES MATIÈRES

INTRODUCTION

L'avènement du domaine de l'intelligence artificielle a conduit des avancés très remarquables dans différents domaines de sciences ; du traitement d'image et la reconnaissance faciale aux robots capables de défier l'intelligence humaine, voire même de la dépasser[1]. Ce concept d'intelligence artificielle reste vague et englobe plusieurs notions dont la plus présente dans la littérature est le *Machine Learning* ou *l'apprentissage automatique*. Il s'agit de développer des algorithmes capables d'apprendre et de créer des modèles statistiques pour l'analyse et la prédiction de données. Les algorithmes d'apprentissage automatique devraient être capables d'apprendre par eux-mêmes, sur la base des données fournies, et de faire des prédictions précises, sans avoir été spécifiquement programmés pour une tâche donnée. Néanmoins, l'entraînement de ces algorithmes nécessite d'avoir en sa disposition un jeu de données assez important selon la complexité du modèle et de la tâche souhaitée.

Le domaine de la recherche en physique n'a pas été épargné par l'utilisation des algorithmes intelligents mais au contraire, il a bénéficié des développements en termes de puissance de calcul des ordinateurs. Si traditionnellement l'expérience était la base de la recherche en physique des matériaux en déterminant les propriétés expérimentales des cristaux, ce qui ne peut être fait qu'à des échelles de temps assez longs et qui se limite à un nombre restreint de matériaux, la première révolution informatique dans la science des matériaux a été alimentée par l'avènement de méthodes informatiques, en particulier la théorie de la fonctionnelle de la densité (DFT), les simulations de Monte Carlo et la dynamique moléculaire, qui ont permis aux chercheurs d'explorer l'espace des phases et de la composition de manière beaucoup plus efficace. Le couplage de ces méthodes de calcul avec les essais expérimentaux ont conduit à la création de bases de données (notamment *MaterialProject*) qui rassemblent plusieurs matériaux avec leurs propriétés, telles que l'énergie de formation, la bande d'énergie interdite (band gap) ..., de telle sorte que l'apprentissage automatique est devenu possible aussi pour les matériaux.

Dans la litterature, on peut trouver une multitude d'applications d'apprentissage automatique dans la science des matériaux, par exemple la prédiction de nouveaux matériaux stables[2], le calcul de nombreuses propriétés des matériaux et l'accélération des calculs de premier principe[3]. Dans la suite de ce travail nous allons construire un modèle de prédiction de l'énergie de formation moyenne des cristaux en se basant sur les données de MaterialProject. Une première difficulté est de faire le pré-traitement des données que nous importerons afin d'avoir des entrées qui sont d'abord vectorielles et puis qui contiennent l'information nécessaire sur le cristal. Ensuite, on choisira le modèle de prédiction parmi ceux présents dans la littérature que nous devons par la suite optimiser pour que les prédictions faites soient les meilleures possibles.

1 CHOIX DE LA BASE DE DONNÉES ET FORMULATION DU PROBLÈME

1.1 Exploration des bases de données

Les consignes étaient de faire des prédictions sur les propriétés physiques d'un matériau à partir de sa structure. En particulier nous avons orienté nos recherches vers le cif et l'énergie (ou enthalpie) de formation moyenne. Le cif décrit toutes les caractéristiques d'une maille d'un cristal : vecteurs de la base de translation grâce à leurs normes et aux angles entre eux, type et position relative de l'ensemble des atomes qui constituent la maille, ainsi que d'autres caractéristiques que nous n'utiliserons pas. L'énergie de formation moyenne est l'énergie qu'il faut fournir pour obtenir le cristal à partir des corps simples, purs, pris dans l'état standard. On divise ensuite par le nombre d'atomes pour avoir une quantité qui ne dépend pas du nombre de mailles dans notre cristal. Cette quantité est alors comparable pour des cristaux qui n'ont pas le même nombre d'atomes par maille. Il est donc pertinent de faire un modèle pour prédire cette quantité.

La formulation du problème que nous avons cherché à résoudre est conditionnée par les données que nous avons à disposition. En effet, sans données à mettre en entrée de notre modèle, impossible d'obtenir des résultats. C'est pourquoi, dans un premier temps, nous avons exploré les bases de données disponibles dans le domaine des matériaux. En voici une liste non exhaustive : MatMatch, Material Project, Material Connexion, ChemSpider, Materials Web, UL Prospector, etc.

MatMatch est une base de données libre d'accès d'environ 40 000 matériaux composites, céramiques, verres, métaux, polymères et matériaux biologiques. Pour chacun de ces matériaux sont proposées un certains nombre de propriétés. Toutefois, elle se concentre sur les propriétés macroscopiques.

Material Project adopte un point de vue beaucoup plus microscopique. Cette base de données propose plus de 140 000 matériaux inorganiques (des matériaux qui ne possèdent pas à la fois une liaison carbone-hydrogène et carbone-carbone). Tous ces matériaux sont une répétition d'une maille élémentaire selon 3 vecteurs de l'espace qui forment une base. Le cif de chaque matériau est disponible dans cette base de données. Et, pour chaque matériaux, le site donne un certain nombre de caractéristiques mécaniques, électriques ou physique : élasticité, bandgap, énergie de formation moyenne, densité...etc. Il faut noter qu'une partie de ces propriétés ne sont pas des résultats expérimentaux mais des prédictions.

Material Connexion recense plus de 10 000 matériaux répartis dans des catégories semblables à MatMatch. Toutefois, le site n'est pas directement libre d'accès. Il faut obligatoirement devenir membre. La procédure est donc plus difficile que les deux bases de données précédentes.

ChemSpider rassemble plus de 100 millions de structures chimiques de toute sorte : molécules, cristaux, etc. C'est un regroupement de nombreuses bases de données. Pour les cristaux, il est notamment possible de récupérer le cif.

1.2 Formulation du problème

Material Project a l'avantage de présenter des méta-données claires. C'est à dire qu'ils expliquent bien l'origine de leurs données, leur volume et leurs caractéristiques notamment. Par ailleurs, cette base de données est assez robuste, surtout pour les propriétés qui vont nous intéresser. Pour tous les matériaux, le cif et l'énergie de formation moyenne sont bien renseignés. De plus, le site propose une bonne documentation de leur API qui nous permettra d'extraire les données.

Ces raisons nous ont conduit à retenir cette base de données. Ainsi l'exploration des données et des travaux dans le domaine ainsi que le respect des consignes nous pousse à formuler le problème de prédiction suivant : nous souhaitons prédire l'énergie de formation moyenne ($formation\_energy\_per\_atom$ sur le site Material Project) à partir du Cif en utilisant les données de Material Project.

1.3 Les étapes de la création d'un modèle de prédiction

Afin de produire un modèle de prédiction en Machine Learning, nous avons identifié un certain nombre d'étapes à suivre. Pour la majorité du groupe, cela constituait notre première expérience dans ce domaine. Dès lors, outre la production de résultats intéressants, nous avons également beaucoup appris sur la méthode de réalisation d'un tel projet.

Voici donc les étapes : La première est celle décrite ci-dessus, c'est une étape d'exploration des bases de données, des modèles déjà existants et d'appropriation des problématiques du domaine sur lequel nous allons travailler, ici les matériaux. Cette exploration permet de formuler un problème, c'est à dire identifier ce qu'on veut prédire, à partir de quelles données. Nous avons choisi de prédire l'énergie de formation moyenne à partir des caractéristiques de la maille élémentaire.

Les étapes suivantes sont : l'extraction des données, la transformation des données, la création de modèles de prédiction et leur comparaison puis l'analyse des résultats. Nous allons décrire ces étapes dans la suite.

2 EXTRACTION ET TRANSFORMATION DES DONNÉES

2.1 EXTRACTION DES DONNÉES DEPUIS MATERIAL PROJECT

Une fois le problème formulé et la base de données choisie, il s'agit d'extraire les données. Pour ce faire, nous avons repéré deux méthodes différentes ; une première qui consiste à utiliser la bibliothèque *json* afin de se connecter mannuellement à la base de donnée de Material Project, et une deuxième façon qui passe par la bibliothèque *matminer* qui est recommandée par Material Project pour faire l'extraction des données à partir de leur base.

- IMPORTATION DES DONNÉES PAR *Json*

En ce qui concerne la première façon de faire, nous avons donc utilisé la bibliothèque *requests* associée à la bibliothèque *json*.

```
13  import json
14  import requests
15
16  # [Pour télécharger les données depuis le site de MaterialsProject]
17  data = {
18      'criteria': {
19          'elements': {'elements': {'$all': ['O']}}
20      },
21      'properties': [
22          'cif',
23          'formation_energy_per_atom',
24          'band_gap'
25      ]
26  }
27  r = requests.post('https://materialsproject.org/rest/v2/query',
28                    headers={'X-API-KEY': '5RfgLbTbu83tNTjO3Vh4'},
29                    data={k: json.dumps(v) for k,v in data.items()})
30  response_content = r.json() # a dict
31
32  # [Pour écrire le resultat dans un fichier sous forme de texte]
33  f = open("dataFormEnCifGrand.txt", "a")
34  f.write(json.dumps(response_content))
35  f.close()
36
37  # [Pour reprendre les données depuis le fichier
38  #  et les mettre sous forme de dict]
39  f = open("dataFormEnCif.txt", "r")
40  truc = json.loads(f.read())
```

FIGURE 1 – Code permettant l'extraction et la sauvegarde des données

Sur la figure ci-dessus, en modifiant le dictionnaire data, il est possible de sélectionner les

matériaux que nous souhaitons récupérer grâce à la valeur associée à la clef *'criteria'*. Ici, *'$all'* signifie qu'on récupère les matériaux qui contiennent au moins un atome de carbone dans la maille élémentaire. Avec *'nin'* on peut exclure les matériaux qui contiennent certains atomes précisés. Et avec *'in'* on exclut tous les matériaux qui contiennent d'autres atomes que ceux spécifiés. Par ailleurs, en modifiant la valeur associée à la clef *'properties'*, on peut sélectionner les propriétés à extraire.

Lien vers la documentation présentant les différentes manières d'extraire les données de Material Project : `https://github.com/materialsproject/mapidoc`

- Importation des données par *Matminer*

Matminer est une bibliothèque open source de Python qui est spécialisé dans le traitement des matériaux physiques afin de pouvoir construire des modèle de Machine Learning. Nous allons utiliser la fonction '*MPDataRetrieval*' qui permet d'accéder directement à la base de donnée de Material Project et d'extraire les matériaux que nous souhaitons étudier en appliquant certains critères.

```
Entrée [3]: from matminer.data_retrieval.retrieve_MP import MPDataRetrieval
            mpdr = MPDataRetrieval(api_key='3g9wMps4VR5Hy6Pm')
            df_Materials = mpdr.get_dataframe(criteria={"elements": {"$in": ['H', 'C']}},
                                              properties=['band_gap', 'formation_energy_per_atom', 'cif'])
            100%|██████████████████████████████████████████████████████████| 13728/13728 [00:30<00:00, 453.42it/s]

Entrée [4]: df_Materials.head()

Out[4]:
```

material_id	band_gap	formation_energy_per_atom	cif
mp-611448	4.4270	0.142597	# generated using pymatgen\ndata_C\n_symmetry_...
mp-1097832	0.0000	2.605766	# generated using pymatgen\ndata_C\n_symmetry_...
mp-1205283	0.1062	0.697407	# generated using pymatgen\ndata_C\n_symmetry_...
mp-24	2.7785	0.833079	# generated using pymatgen\ndata_C\n_symmetry_...
mp-1078845	3.5883	0.265893	# generated using pymatgen\ndata_C\n_symmetry_...

Figure 2 – Code permettant l'extraction et la sauvegarde des données par Matminer

Dans la figure 2, on a extrait les matériaux contenant à la fois les atomes de l'hydrogène et du carbone (ce qui fait presque 14000 matériaux). On récupère alors un DataFrame dont les colonnes sont les propriétés que nous avons fixées lors de l'appel de la fonction '*MPDataRetrieval*', ce qui constitue en effet l'avantage de cette approche. On manipulera ainsi dans la suite un DataFrame, plus pratique à utiliser en entrée des modèles de Machine Learning.

2.2 Transformation des données

Lors de l'élaboration d'un modèle de prédiction, nous avons découvert qu'il faut trouver un bon compromis entre transformation des données et complexité du modèle. En effet, prédire l'énergie de formation moyenne directement à partir de la position des atomes dans la

maille et des paramètres de mailles va nécessiter un modèle très complexe avec énormément de paramètres et donc très long à entraîner. Si, dans l'excès inverse, on calcule tout avec les modèles physiques, il n'y a plus de Machine Learning et on ne peut espérer faire mieux que les approximations de ces modèles physiques. Tout l'intérêt du Machine Learning réside dans l'espoir que l'optimisation des paramètres d'un modèle assez général va nous permettre d'approcher plus finement la vraie fonction (sous l'hypothèse épistémologique qu'elle existe) que les modèles physiques approchés déjà existant.

Dès lors, l'objectif est de transformer les données relativement simplement en utilisant quelques notions de physique, pour ensuite créer un modèle de prédiction qui prend en entrée ces nouvelles données transformées.

Dans un premier temps, nous avons cherché à transformer les données à la main. Nos connaissances physiques nous laissent penser que l'énergie de formation moyenne dépend essentiellement de la distance entre les différents atomes d'une maille et de leur nature. Dès lors nous souhaitions calculer l'ensemble des distances entre types d'atomes et les ranger dans une matrice de taille 83 par 83, par exemple, si nous considérons l'ensemble des 83 premiers atomes du tableau périodique. En effet, il ne faut pas oublier que nos modèles de prédiction prennent en entrée un ensemble de $features$ de taille fixe. Deux problèmes se posent alors :

Tout d'abord, il ne faut pas seulement calculer les distances entre les atomes de la maille élémentaire, il faut prendre en compte les interactions entre mailles. Nous avons donc imaginé prendre une distance maximum de calcul, par exemple 30 Angströms et ne prendre en compte que les distances entre atomes inférieures à cette valeur, les autres étant supposées négligeables dans le calcul de l'énergie de formation moyenne.

En outre, on s'accordera à dire que "ranger" les "distances" dans une matrice de taille 83 par 83 est une formulation vague. L'objectif est d'obtenir une énergie de formation moyenne, il va donc falloir diviser par le nombre d'atomes ou de distances considérés quelque part. Par ailleurs, dans une case de la matrice [i,j] on peut par exemple sommer les distances entre les atomes i et les atomes j. Puis diviser l'ensemble de la matrice par le nombre de distances considérées. Ou bien sommer les inverses des distances, ou bien faire des moyennes dans chaque case. Ce sont autant d'idées qu'il aurait été intéressant de tester en les prenant comme entrées d'un modèle et en observant si on obtient de bonnes prédictions de l'énergie de formation moyenne.

Nous travaillions en parallèle sur des représentations des données, déjà implémentées, que nous aurions pu utiliser à la place des calculs envisagés dans les paragraphes précédents.

Cette approche a été fructueuse et nous avons trouvé deux bibliothèques qui conjointement permettent d'obtenir des représentations pertinentes des données à partir des caractéristiques de la maille, afin de prédire l'énergie de formation moyenne. Ces bibliothèques sont *ase* et *dscribe*. *ase* propose un type *Atoms* qui stocke essentiellement les mêmes données que le cif mais qui permet également de visualiser les mailles élémentaires, ce qui est très utiles pour vérifier que le code ne comporte pas d'erreur et pour mieux s'approprier la physique qui se

cache derrière le calcul de l'énergie de formation. Le type *Atoms* est surtout utile car c'est ce format que les fonctions de *dscribe* prennent en entrée afin de transformer les données.

La bibliothèques *dscribe* met à disposition un certain nombre de représentations : le Smooth Overlap of Atomic Positions (SOAP), la matrice de Coulomb, la Many-body Tensor Representation (MBTR) ou encore la Atom-centered Symmetry Function (ACSF). Nous avons principalement utilisé le *SOAP* et la matrice de Coulomb. [1]

2.2.1 • Extraction du SOAP

Nous présentons dans ce qui suit l'ensemble des fonctions qui permettent de passer du *cif* au *SOAP*. Tout d'abord, il faut créer une classe SOAP.

```
7   from pymatgen.io.cif import CifParser
8
9   from dscribe.descriptors import SOAP
10  #from ase.build import bulk
11  from ase import Atoms
12  from ase.visualize import view
13  import scipy
```

Figure 3 – Importation nécessaire à la transformation des données en SOAP

```
20  #==========# Constantes #==========#
21
22  species = ['H','Li','B','C','N','O','F','Na','Mg','Al','Si',
23             'P','S','Cl','K','Ca','Ti','V','Cr','Mn','Fe','Co',
24             'Ni','Cu','Zn','Se','Sr','Ba']
25  rcut = 6.0
26  nmax = 3
27  lmax = 2
28
29  dictionnaireTabPeriodique = {'H' : 1, 'He' : 2, 'Li' : 3, 'Be' : 4,
59
60  #==========# Création classe SOAP #==========#
61
62  print('creation of the SOAP class ...')
63
64  periodic_soap = SOAP(
65      species=species,
66      rcut=rcut,
67      nmax=nmax,
68      lmax=lmax,
69      periodic=True,
70      sparse=False #format COO
71  )
72
73  print('SOAP created')
```

Figure 4 – Création du SOAP

1. Plus de détails : `https://www.sciencedirect.com/science/article/pii/S0010465519303042`

Les constantes sur la Figure 4 permettent de fixer la précision de notre nouvelle représentation des données qui va être la même pour tous les *instances*, c'est à dire pour toutes les mailles d'entraînement et de test. Cela permet d'avoir un nombre de $features$ constant pour toutes les *instances*. Dès lors, on ne pourra pas avoir des mailles qui contiennent d'autres atomes que ceux spécifiées dans *species*. Plus *nmax* et *lmax* sont grands, plus on va aller loin dans le calcul des harmoniques sphériques et des fonctions de la base radial.

```
77  def extractCifDict(i):
78      '''
79      Parameters
80      ----------
81      i : int
82          Indice entre 0 et len(baseDonnee['response']).
83
84      Returns
85      -------
86      instanceCifDict : dict
87          Contient le cif de l'instance i sous forme de dictionnaire.
88
89      '''
90      instanceCifPars = CifParser.from_string(baseDonnee['response'][i]['cif'])
91      instanceCifDict = instanceCifPars.as_dict()
92      instanceCifDict = instanceCifDict.popitem()[1]
93      return instanceCifDict
```

FIGURE 5 – Transformation d'un cif en dictionnaire python

```
95  def computeMaille(instanceCifDict):
96      '''
97      Parameters
98      ----------
99      instanceCifDict : dict
100         Contient le cif de l'instance i sous forme de dictionnaire.
101
102     Returns
103     -------
104     maille : ase.Atoms
105         La maille associée au cif au format ase.Atoms.
106
107     '''
108     #Constantes
109     a = float(instanceCifDict['_cell_length_a'])
110     b = float(instanceCifDict['_cell_length_b'])
111     c = float(instanceCifDict['_cell_length_c'])
112     alpha = float(instanceCifDict['_cell_angle_alpha'])
113     beta = float(instanceCifDict['_cell_angle_beta'])
114     gamma = float(instanceCifDict['_cell_angle_gamma'])
115     #Cell
116     in_cell = [a, b, c, alpha, beta, gamma]
117     #Numbers
118     in_numbers = [dictionnaireTabPeriodique[i]
119                   for i in instanceCifDict['_atom_site_type_symbol']]
120     #Positions
121     tabX = np.array(instanceCifDict['_atom_site_fract_x']).astype(float)
122     tabY = np.array(instanceCifDict['_atom_site_fract_y']).astype(float)
123     tabZ = np.array(instanceCifDict['_atom_site_fract_z']).astype(float)
124     tabX = tabX.reshape((1,tabX.shape[0]))
125     tabY = tabY.reshape((1,tabY.shape[0]))
126     tabZ = tabZ.reshape((1,tabZ.shape[0]))
127     in_positions = np.concatenate((tabX, tabY, tabZ),axis = 0).transpose()
128     #Construction maille avec ase
129     maille = Atoms(cell = in_cell,
130                    numbers = in_numbers,
131                    positions = np.zeros(in_positions.shape),
132                    pbc=[True, True, True])
133     maille.set_scaled_positions(in_positions)
134     return maille
```

FIGURE 6 – Transformation d'un dictionnaire python en objet de la classe *Atoms*

La Figure 6 présente l'extraction de toute les données du cif pour créer l'objet de la classe *Atoms* correspondant. Dans les arguments fournis au constructeur de la classe *Atoms*, on remarque que *pbc* prend la valeur $[True, True, True]$. Cela signifie que la maille se répète à l'infini selon les trois vecteurs de maille a, b et c de *in_cell*. L'avantage de cette représentation est qu'il existe une méthode toute faite *set_scaled_positions*() qui convertit les positions relatives des atomes dans la maille en positions absolues dans la base cartésienne. Le *cif* donne les positions relatives tandis que le constructeur d'*Atoms* prend les positions absolues. Dans notre première approche, à la main, nous avions calculé la matrice de changement de base en fonction des paramètres de maille (a, b, c, α, β, γ). Ce calcul n'est pas aisé, il s'agit d'une orthonormalisation de Gram-Schmidt en dimension trois à partir d'une paramétrisation inhabituelle mais il était nécessaire pour calculer les distances cartésiennes.

```
137  def computeSoapReduced(maille):
138      '''
139      Parameters
140      ----------
141      maille : ase.Atoms
142          La maille associée au cif au format ase.Atoms.
143
144      Returns
145      -------
146      soap_maille : scipy.sparse.coo_matrix
147          Tableau numpy de shape (1, nbFeatures) transformé en sparse Matrix.
148          Avec nbFeatures qui dépend de la construction de notre instance
149          de SOAP.
150
151      '''
152      soap_maille = periodic_soap.create(maille)
153      nbAtomes = len(instanceCifDict['_atom_site_fract_x'])
154      soap_maille = soap_maille.sum(axis=0).reshape(1, -1) / nbAtomes
155      #soap_maille = scipy.sparse.coo_matrix(soap_maille)
156      return soap_maille
```

Figure 7 – Transformation d'un objet de la classe *Atoms* en SOAP

Nous obtenons enfin le SOAP de notre atome. En fonction de la taille de la base de données qu'on souhaite convertir et des précisions choisies (c'est-à-dire des valeurs de $nmax$ et $lmax$), nous avons trouvé judicieux de sauvegarder cela sous forme de $sparseMatrix$ grâce aux classes *sparse.coo_matrix* et *sparse.csr_matrix* de la bibliothèques *scipy*. En effet, nous nous retrouvions vite avec des tableaux numpy de tailles de l'ordre de quelques gigaoctets alors que la plupart des valeurs stockées sont nulles. La classe de matrice *sparse.csr_matrix* permet de faire facilement du slicing sur les lignes ce qui facilite le découpage entre instances d'entraînement et instances de test.

2.2.2 • Extraction de la matrice de Coulomb

Comme on vient de le voir avec l'objet 'SOAP' de la bibliothèque *dscribe*, on trouve l'objet 'CoulombMatrix' dont le fonctionnement est similaire. Si les paramètres nécessaires pour définir cet objet sont différents de ceux avec lesquels on définit 'SOAP', ils ont tous les deux la méthode

'*create*' qui transforme un objet *Atoms* en une matrice de Coulomb ou en ses valeurs propres. La matrice de Coulomb se définit comme :

$$M_{ij}^{\text{Coulomb}} = \begin{cases} 0.5 Z_i^{2.4} & \text{for } i = j \\ \frac{Z_i Z_j}{R_{ij}} & \text{for } i \neq j \end{cases} \tag{1}$$

Sa dimension est alors $n \times n$ pour un matériau contenant n atomes dans sa maille centrale. Pour instancier donc l'objet 'CoulombMatrix', nous devons préciser un N_{max} qui sera la dimension maximale de la matrice. De plus, pour avoir l'invariance par permutation des atomes de la maille d'un matériau, nous allons renvoyer les valeurs propres de la matrice de Coulomb mises en ordre croissant. La fonction 'CoulombMatrix.create' va renvoyer alors un vecteur de dimension N_{max}

```
Entrée [14]: N_max = max([len(at.get_positions()) for at in atoms])
             print(N_max)

             444

Entrée [15]: Cm = CoulombMatrix(n_atoms_max=N_max,
                                 permutation='eigenspectrum')
             print(atoms[0])
             Cm.create(atoms[0])

             Atoms(symbols='C12', pbc=True, cell=[[2.52225173, 0.0, 0.0], [-1.2611255992769543, 2.1843342263344994, 0.0], [0.0, 0.0, 12.4
             33927]])

Out[15]:     array([156.92499378,  79.53621847,  51.73484093,  36.42771401,
                     28.81365293,  23.02062919,  15.74181861,  14.07525967,
                     12.39232677,   9.72048497,   7.48333231,   6.42599076,
                      0.        ,   0.        ,   0.        ,   0.        ,
                      0.        ,   0.        ,   0.        ,   0.        ,
                      0.        ,   0.        ,   0.        ,   0.        ,
                      0.        ,   0.        ,   0.        ,   0.        ,
                      0.        ,   0.        ,   0.        ,   0.        ,
                      0.        ,   0.        ,   0.        ,   0.        ,
                      0.        ,   0.        ,   0.        ,   0.        ,
                      0.        ,   0.        ,   0.        ,   0.        ,
                      0.        ,   0.        ,   0.        ,   0.        ,
                      0.        ,   0.        ,   0.        ,   0.        ,
                      0.        ,   0.        ,   0.        ,   0.        ,
                      0.        ,   0.        ,   0.        ,   0.        ,
                      0.        ,   0.        ,   0.        ,   0.        ,
```

Figure 8 – Transformation d'un objet de la classe *Atoms* : C_{12} en le vecteur de valeurs propres de la matrice de Coulomb associée

La matrice de Coulomb calcule en effet d'une façon approximative le potentiel électrostatique appliqué sur chacun des atomes constituant le matériau. On pourrait donc espérer que ça soit une grandeur explicative de l'énergie pour le modèle de prédiction.

3
MODÈLES DE PRÉDICTION

Nous disposons désormais de deux outils permettant de représenter un matériau sous une forme vectorielle, nécessaire pour l'utilisation des algorithmes de Machine Learning. On fera en premier lieu un modèle de prédiction sur un jeu de données petit en ne prenant que les cristaux qui contiennent du carbone afin d'avoir une idée sur la performance des algorithmes de ML dans ce contexte de la physique des matériaux.

On élargira dans un second temps le jeu de données en prenant les matériaux dont les numéros atomiques de leurs atomes ne dépassent pas une valeur maximale Z_{max} qu'on fixera à 40, C'est à dire que nous considérerons que les matériaux dont les atomes se localisent sur les quatre premières colonnes du tableau périodique des éléments.

3.1 Prédiction sur les cristaux contenant du carbone

On extrait alors de la base de données de Material Project tous les matériaux contenant du Carbone, en particulier on extrait les CIF ainsi que les énergies de formation par atome. Au total, on récupère 7603 matériaux. Pour simplifier encore le modèle, nous allons procéder à un filtre pour ne considérer que les matériaux dont le numéro atomique des atomes qui les constituent ne dépasse pas $Z_{max} = 56$. Ainsi, on conservera au final 5304 mailles.

```
Zmax = 56
rcut = 6.0
nmax = 5
lmax = 5


periodic_soap = SOAP(
    species=[i for i in range(1,Zmax+1)],
    rcut=rcut,
    nmax=nmax,
    lmax=lmax,
    periodic=True,
    sparse=False
)
```

Figure 9 – Définition de l'objet SOAP

En fixant les paramètres de l'objet $SOAP$ comme montré sur la figure ci-dessus, nous obtenons pour chaque matériau un SOAP de taille 236040, c'est à dire que nous disposons de plus de 200000$features$ pour entrainer le modèle de prédiction. On résume le jeu de données sur ce tableau :

Matériaux avec uniquement du Carbone	Dimension des SOAP	Valeur minimale du SOAP	valeur maximale du SOAP
Train set	(5000, 236040)	−11762.60	28178.21
Test set	(304, 236040)	−1742.11	4319.65

On remarque donc que l'intervalle de variation des SOAP est assez grand ; c'est de l'ordre de 10^4, une normalisation des SOAP pourrait être alors nécessaire pour que le modèle de régression ne soit pas biaisé par cette grande échelle de variation. Pour vérifier si l'échelle aura une importance dans ce cas, nous allons entraîner deux modèle de régression avec des réseaux de neurones à 4 couches denses.

- **Modèle de régression avec des réseaux de neurones**

Nous allons utiliser la bibliothèque 'Keras' de python qui fournit des outils pour construire un réseau de neurones. Vu la grande dimension des $features$ (en l'occurrence $236040 features$), on va utiliser un réseau de neurones à 4 couches dont les nombres des unités sont $1048, 256, 128$ et 64. Au total nous aurons plus de 200 millions de paramètres à optimiser :

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1048)              247370968

 dense_1 (Dense)             (None, 256)               268544

 dense_2 (Dense)             (None, 128)               32896

 dense_3 (Dense)             (None, 64)                8256

 dense_4 (Dense)             (None, 1)                 65

=================================================================
Total params: 247,680,729
Trainable params: 247,680,729
Non-trainable params: 0
_________________________________________________________________
```

Figure 10 – Résumé du réseau de neurone

On procédera ensuite au fitting du modèle avec les données qui sont à notre disposition, on obtient alors la courbe qui montre la variation de la fontion de perte (MAE) tout au long de la phase d'apprentissage. Il est à noter qu'on réalisera deux modèles différents ; un premier qui prendra comme entrée les données non normalisées et un autre modèle qui récupérera en entrée les SOAP normalisés.

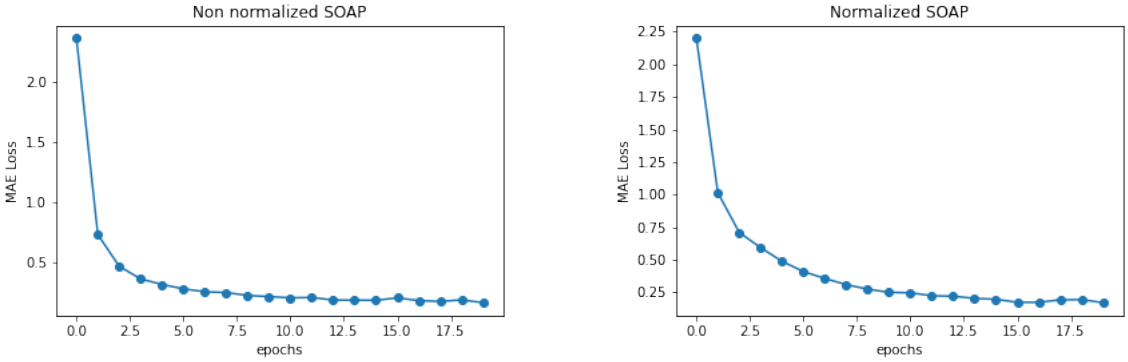


Figure 11 – Phase d'apprentissage des deux modèles

On remarque alors que l'apprentissage, c'est-à-dire la pente de décroissance de la courbe des pertes, est plus rapide avec les données sans normalisation. On procédera alors au test sur le reste des données pour voir la performance du modèle :

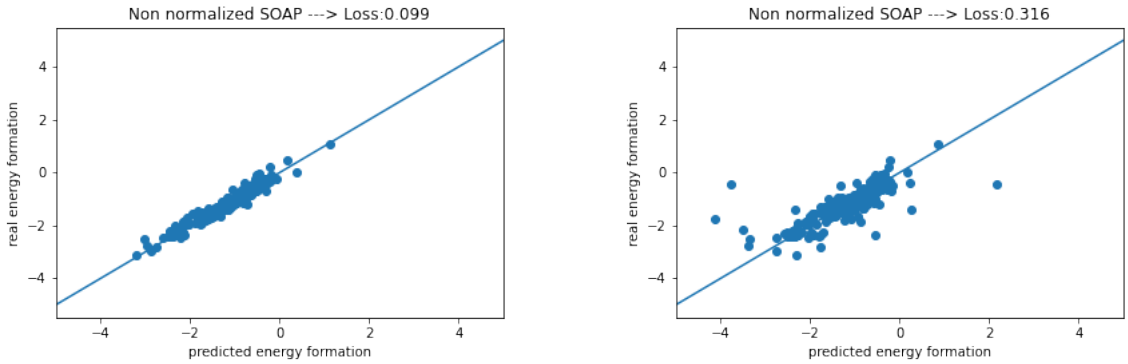


Figure 12 – Phase d'apprentissage des deux modèles

En se basant sur le test réalisé sur les deux modèles, on constate d'abord qu'ils ont tous les deux des pertes négligeables avec plus d'efficacité pour le modèle avec les données non normalisées, une perte qui est de l'ordre de 10^{-2}. Ainsi, la normalisation des données d'entrées, ici les SOAP, ne contribue pas à la perfection du modèle mais plutôt à une faiblesse quant à la précision des nouvelles prédictions. On gardera alors pour la suite les données comme elles sont sans faire une normalisation.

On vient de voir donc un premier modèle avec un jeu de données réduit, et la performance que nous avons constaté vers la fin était correcte et satisfaisante. Mais, il reste à voir la performance d'un tel modèle sur un jeu de données plus grand, et c'est ce que nous allons faire dans ce qui suit.

3.2 Prédiction sur un jeu de données plus grand

Dans cette partie, nous allons nous intéresser à la prédiction de l'energie de formation par atome des matériaux dont les numéros atomiques de leurs atomes ne dépassent pas une valeur maximale $Z_{max} = 40$. On a alors récupéré un ensemble de 47743 matériaux. On va mettre en place des modèles de prédiction sur le SOAP et d'autres sur les valeurs propres de la matrice de Coulomb. On comparera ainsi vers la fin la performance des deux approches.

- **Modèles avec le SOAP**

On va considérer l'objet SOAP suivant :

```
Entrée [4]: Zmax = 40
            rcut = 6.0
            nmax = 8
            lmax = 6


            periodic_soap = SOAP(
                species=[i for i in range(1,Zmax+1)],
                rcut=rcut,
                nmax=5,
                lmax=5,
                periodic=True,
                sparse=False
            )
```

Figure 13 – Définition de l'objet SOAP plus large

Nous avons pris des valeurs de n_{max} et de l_{max} plus grandes afin d'espérer avoir un jeu de données qui va nous faciliter la prédiction. Vu qu'on a mis un Z_{max} plus petit par rapport au cas du Carbone, et même si on a élargie les autres paramètres, on obtient $120601 features$ dans le SOAP.

On a voulu faire le fitting du modèle de prédiction directement sur l'ensemble des SOAP disponibles, une entrée de dimension $(47743, 120601)$, mais malheureusement les ordinateurs que nous utilisons n'ont pas pu supporter ce nombre gigantesque d'opérations et le noyau de Python finit par crasher à cause de la saturation de la RAM. De ce fait, on a décidé de faire l'apprentissage uniquement sur la moitié des SOAP disponibles, c'est-à-dire 20000 SOAP :

Matériaux avec un $Z_{max} = 40$	Dimension des SOAP	Valeur minimale du SOAP	valeur maximale du SOAP
Train set	(20000, 120601)	-11762.60	28178.21
Test set	(200, 120601)	-1240.85	2997.07

On a travaillé avec deux réseaux de neurones différents, un premier avec 4 couches denses comme ce qu'on a fait avec les matériaux contenant du Carbone, et un deuxième avec une couche de plus pour voir à quel point le modèle sera amélioré. Néanmoins, il faut faire en sorte que le modèle ne soit pas dans le cas du sur-apprentissage (over-fitting) pour que la prédiction

Model: "sequential"

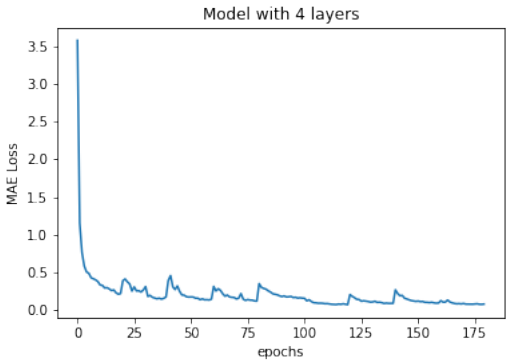
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1048)	126389848
dense_1 (Dense)	(None, 256)	268544
dense_2 (Dense)	(None, 128)	32896
dense_3 (Dense)	(None, 64)	8256
dense_4 (Dense)	(None, 1)	65

Total params: 126,699,609
Trainable params: 126,699,609
Non-trainable params: 0

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1048)	126389848
dense_1 (Dense)	(None, 1048)	1099352
dense_2 (Dense)	(None, 256)	268544
dense_3 (Dense)	(None, 128)	32896
dense_4 (Dense)	(None, 64)	8256
dense_5 (Dense)	(None, 1)	65

Total params: 127,798,961
Trainable params: 127,798,961
Non-trainable params: 0



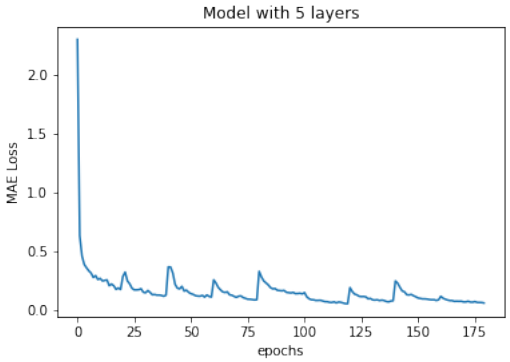
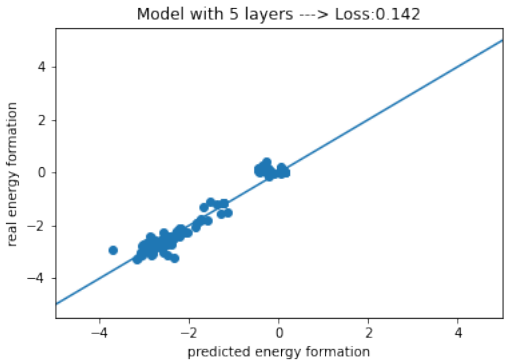


Figure 14 – Les deux modèles du réseau de neurones avec les courbes d'apprentissage

soit bonne, et dans ce cas l'augmentation du nombre des couches cachées pourrait entraîner un sur-apprentissage.

Le modèle à 5 couches a presque 1 million de paramètres à optimiser de plus que le modèle à 4 couches, ce qui fait qu'il prend plus de temps dans la phase de l'apprentissage. Néanmoins, les valeurs de pertes atteintes sont très comparables pour les deux modèles. Afin de trancher entre les deux modèles, il faut alors voir leurs performances sur l'ensemble de test.



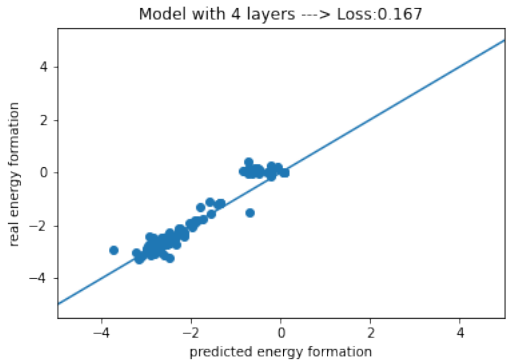


Figure 15 – Phase d'apprentissage des deux modèles

On constate alors que la prédiction sur les deux modèles est satisfaisante, même si le modèle à cinq couches est relativement meilleur. Mais, compte tenu du temps d'apprentissage moins

important pour ce dernier, il sera préférable tant qu'il reproduit des résultats aussi bon que celui avec cinq couches. Enfin, les modèles de prédiction pour l'énergie de formation par atome avec le SOAP donnent des résultats assez satisfaisants et dont la performance dépend des classes des matériaux qu'on considère. Ainsi, on pourrait conclure sur le caractère explicatif de cette énergie de formation par le SOAP. Qu'en est-il de la prédiction moyennant la matrice de Coulomb ? C'est ce qu'on va voir dans la partie suivante.

• Modèle avec les valeurs propres de la matrice de Coulomb

On reprend les 47743 matériaux que nous avons récupérés auparavant et on va construire un réseau de neurone qui prend en entrée cette fois-ci les valeurs propres de la matrice de Coulomb pris dans l'ordre croissant de chaque matériau :

```
Entrée [4]: Nel_max = max([len(at.get_positions()) for at in atoms])
            print(Nel_max)

            432

Entrée [5]: cm = CoulombMatrix(n_atoms_max=Nel_max, permutation='eigenspectrum')
```

Figure 16 – Définition de l'objet CoulombMatrix

Puisque le N_{max} ici est égal à 432, nous aurons $432 features$ pour décrire chaque matériau. Ainsi, la dimension des features n'est pas grande de telle façon qu'on peut prendre tout le jeu de données à la fois pour faire l'apprentissage et le test du modèle, on prend alors :

Matériaux avec un $Z_{max} = 40$	Dimension des valeurs propres	Valeur minimale	valeur maximale
Train set	(45000, 432)	0.0	28178.21
Test set	(300, 432)	−1.31	21599.00

On passe alors directement à la construction du modèle, on prendra ici encore deux modèle ; un premier à quatre couches comme on a fait avec le SOAP et un autre en essayant d'enlever une couche puisque la nombres des $features$ est plus petit. On procède ensuite à l'entraînement du modèle.

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_23 (Dense)            (None, 256)               110848

 dense_24 (Dense)            (None, 128)               32896

 dense_25 (Dense)            (None, 64)                8256

 dense_26 (Dense)            (None, 1)                 65

=================================================================
Total params: 152,065
Trainable params: 152,065
Non-trainable params: 0
_________________________________________________________________
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1048)              453784

 dense_1 (Dense)             (None, 256)               268544

 dense_2 (Dense)             (None, 128)               32896

 dense_3 (Dense)             (None, 64)                8256

 dense_4 (Dense)             (None, 1)                 65

=================================================================
Total params: 763,545
Trainable params: 763,545
Non-trainable params: 0
_________________________________________________________________
```

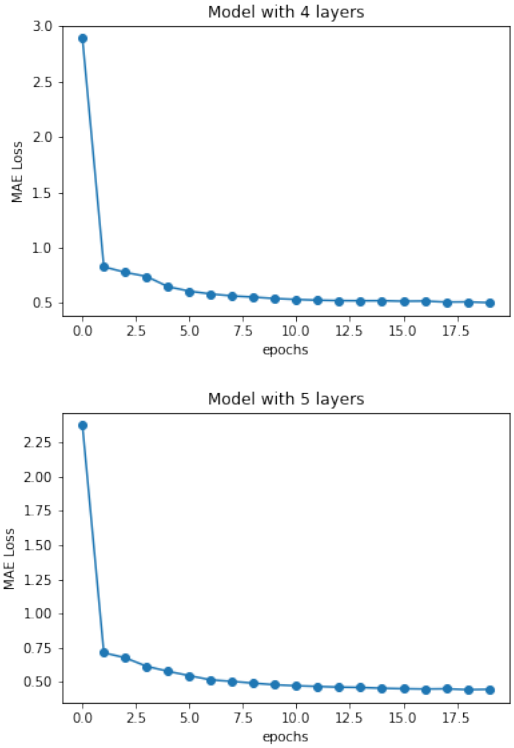


Figure 17 – Les deux modèles du réseau de neurones avec les courbes d'apprentissage

Ce qui est remarquable ici par rapport aux modèles avec le SOAP est le nombre total des paramètres du modèle, on descend de plus de quatre ordres de grandeurs, ce qui réduit très significativement le temps entraînement du modèle. On passe donc à la phase de test pour évaluer le modèle :

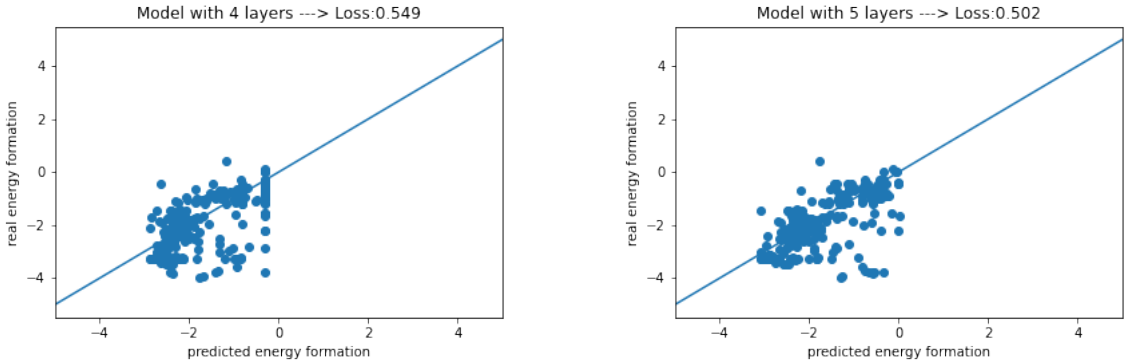


Figure 18 – Phase d'apprentissage des deux modèles

On trouve donc que les deux modèles ont presque la même valeur du *Loss*, par contre on voit bien que le modèle à 4 couches a mis une même valeur pour plusieurs observations (la colonne qui est proche de l'abscisse $x = 0$), ce qui signifie que ce modèle est incapable de faire la différence entre ces matériaux. Ainsi, le choix du modèle à cinq couches est plus pertinent.

• Modèle de régression avec des réseaux de neurones convolutifs

On va exploiter le fait que notre descripteur d'atomes (la liste triée des valeurs propres de la matrice de Coulomb) est invariant par rotation et translation pour pouvoir utiliser les réseaux de neurones convolutifs (CNN). Ce type de réseaux est particulièrement réservé pour les images mais l'invariance par rotation en chimie, le rend parfaitement adapté pour notre étude. Nous utiliserons les réseaux de neurones convolutifs $1D$ et $2D$. Ces réseaux sont généralement utilisés lorsqu'il s'agit de séries temporelles (séquences), ce qui est bien le cas ici, dans le cas 2D on va redimensionner la matrice d'entrée CMs.

Réseau de neurones convolutifs $1D$: Pour le jeux de données on exploitera 0.9 des 47 743 matériaux pour la phase d'apprentissage tandis que le reste va être réservé pour le test puis on va utiliser une structure de réseaux à 6 couches vu le nombre élevé de données qu'on possède et on prendra comme paramètres ceux qui sont résumés dans la figure ci-dessous, ils nous ont permis d'avoir plus de 300,000 paramètres d'entraînement . Ce qui entraîne un temps d'apprentissage légèrement élevé.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv1d_28 (Conv1D)           (None, 423, 32)           352

 conv1d_29 (Conv1D)           (None, 417, 128)          28800

 conv1d_30 (Conv1D)           (None, 413, 128)          82048

 conv1d_31 (Conv1D)           (None, 411, 256)          98560

 flatten_9 (Flatten)          (None, 105216)            0

 dense_11 (Dense)             (None, 1)                 105217

=================================================================
Total params: 314,977
Trainable params: 314,977
Non-trainable params: 0
```

Figure 19 – Résumé CNN 1D

On retrouve les courbes du MAE en fonction de l'epoch dans la figure ci-dessous pour les deux données : train et test.

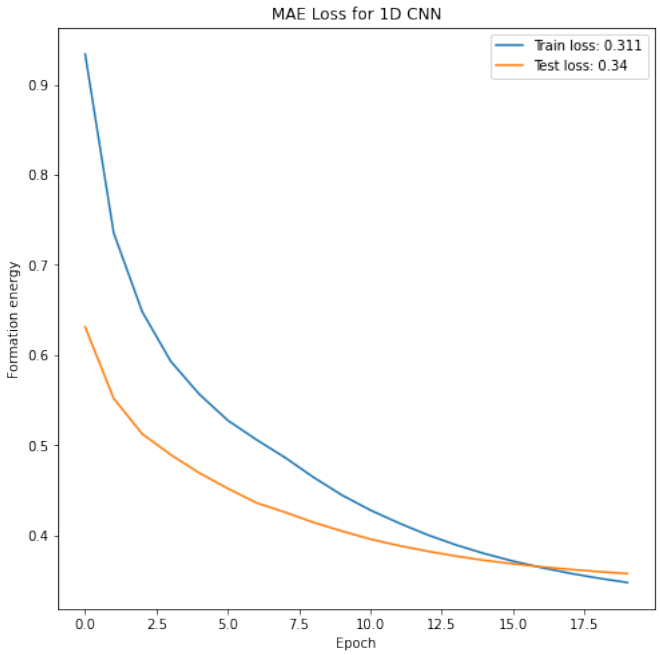
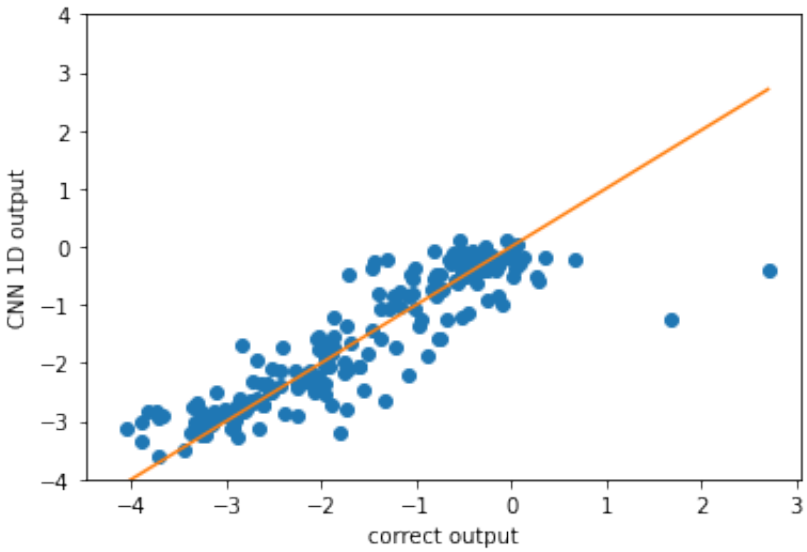


Figure 20 – MAE pour CNN 1D

On remarque que la MAE décroît rapidement avec l'epoch , donc pour avoir de meilleurs résultats il suffit d'augmenter le nombre de training.

Le modèle de prédiction nous a permis d'obtenir un estimation de l'énergie avec une MAE de l'ordre de 0.3 ce qui est acceptable vue que l'énergie varie dans l'intervalle $[-4, 4]$ qui est de longueur 8. Voici ci-dessus la représentation des données estimées en fonction des données réelles pour 200 matériaux.



Réseau de neurones convolutifs $2D$: On commence d'abord par redimensionner le jeux de données afin d'avoir comme entrée des matrices carrées au lieu d'une liste, on sera contraint dans ce cas à augmenter le N_{max} afin que N_{max} soit un carré parfait, ceci est équivalent à compléter par des features nulles et cela ne va pas affecter le jeux de données. Afin de comparer objectivement entre les deux modèles $1D$ et $2D$ nous utiliserons le même split de données, c'est-à-dire 0.9 pour le training et le reste pour le test. Puis on utilisera une structure de réseaux à 8 couches au vu du nombre élevé de données qu'on possède, on prendra comme paramètres ceux qui sont résumés dans la figure ci-dessous, ils nous ont permis d'avoir plus de 350,000 paramètres d'entraînement .Ce qui entraîne comme dans le modèle précédent un temps d'apprentissage assez élevé.

Layer (type)	Output Shape	Param #
conv2d_118 (Conv2D)	(None, 23, 23, 32)	1600
conv2d_119 (Conv2D)	(None, 19, 19, 64)	51264
conv2d_120 (Conv2D)	(None, 17, 17, 64)	36928
conv2d_121 (Conv2D)	(None, 15, 15, 64)	36928
conv2d_122 (Conv2D)	(None, 13, 13, 128)	73856
conv2d_123 (Conv2D)	(None, 11, 11, 128)	147584
flatten_33 (Flatten)	(None, 15488)	0
dense_35 (Dense)	(None, 1)	15489

Total params: 363,649
Trainable params: 363,649
Non-trainable params: 0

Figure 21 – Résumé CNN 2D

On retrouve les courbes du MAE en fonction de l'epoch dans la figure ci-dessus pour les deux jeux de données : train et test.

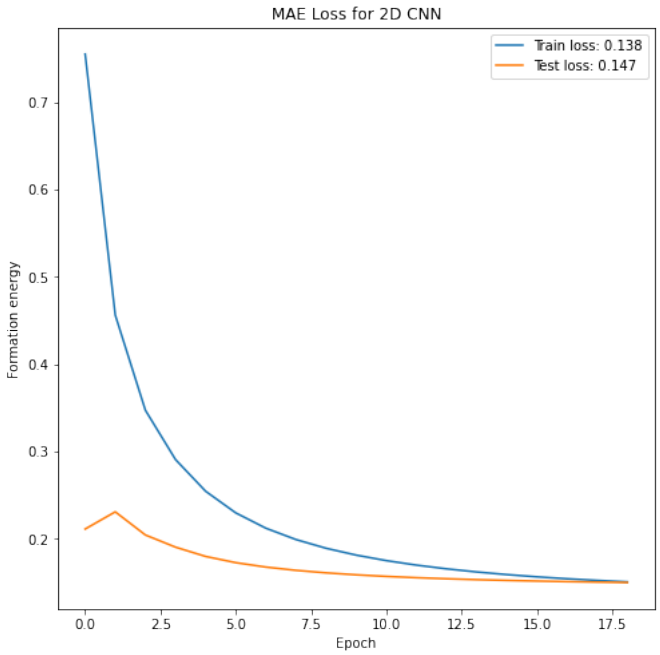
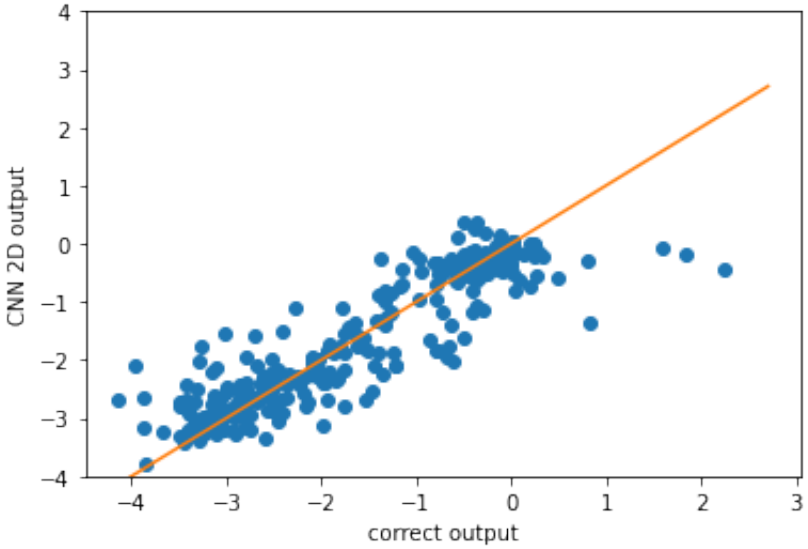


Figure 22 – MAE pour CNN 2D

On remarque que la MAE décroît avec l'epoch mais plus rapidement que dans le modèle 1D , donc pour avoir de meilleurs résultats il suffit d'augmenter plus légèrement le nombre de training que dans le modèle précédent.

Le modèle de prédiction nous a permis d'obtenir une estimation de l'énergie avec une MAE de l'ordre de 0.1 ce qui est bien inférieur que dans le modèle précèdent. Voici ci-dessus la représentation des données estimées en fonction des données réelles pour 200 matériaux.



Comparaison des deux modèles : Après l'analyse des résultats obtenus dans les deux modèles, on pourra dire que comme le nombre de paramètres à entraîner dans les deux cas est approximativement pareil,les temps d'apprentissage le seront aussi. La différence notable sur le plan des performances entre les deux modèles se situent au niveau des erreurs absolues moyennes qui dans le cas du modèle $1D$ 3 fois plus grande que dans le modèle $2D$. Les allures des représentations des données estimées en fonction des données réelles semblent être semblables avec une distance des points estimés par rapport à la médiane plus grande dans le cas $1D$ ce qui est cohérent avec le fait qu'on ait une erreur plus grande dans ce modèle par rapport au modèle $2D$. Dans les deux modèles on remarque, une difficulté à estimer dans le cas des matériaux à énergie élevée.

- ## Modèle de régression avec XGBoost

XGBoost est un algorithme d'apprentissage machine basé sur un arbre de décision d'ensemble et utilise un cadre de renforcement de gradient. Les réseaux neuronaux artificiels sont généralement plus performants que d'autres cadres ou algorithmes lorsqu'il s'agit de prédire des problèmes avec le texte, les images et d'autres données non structurées.

Les choses sont très différentes lorsqu'il s'agit de données tabulaires/structurées de taille moyenne ou petite parce que les algorithmes basés sur des arbres de décision sont sans doute la meilleure solution et sont plus performants.

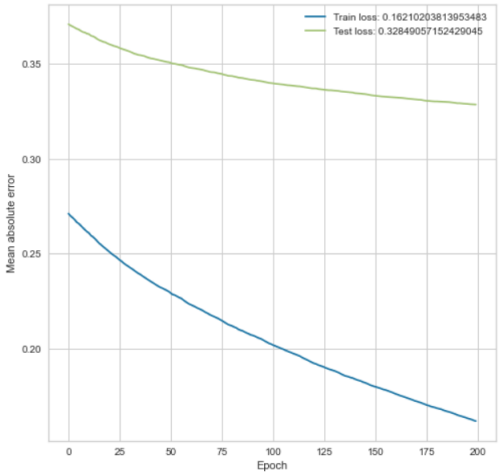
Le Boosting : Le Boosting est une procédure alternative où chaque enquêteur modifie les critères d'évaluation en fonction des commentaires reçus des enquêteurs précédents. Elle permet un processus d'entretien très efficace car elle contribue à déployer des procédures d'évaluation plus puissantes.

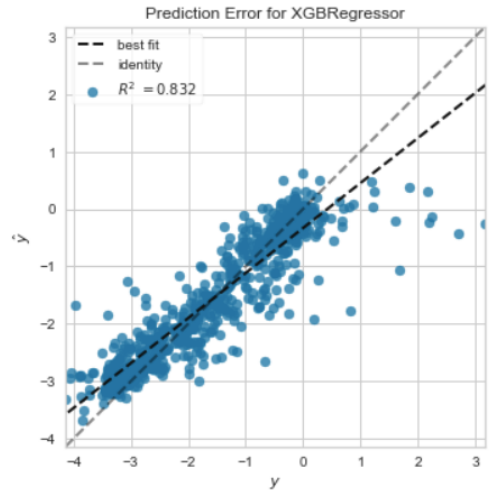
Gradient Boosting : Le processus de renforcement du gradient minimise les erreurs grâce à l'algorithme de descente du gradient. Par exemple, les sociétés de conseil en stratégie utilisent les entretiens de cas pour éliminer les candidats qui ne sont pas aussi qualifiés que les autres candidats.

XGBoost : Le XGBoosting est une version nettement plus efficace du gradient boosting, d'où son nom : Extreme Gradient Boosting. C'est une combinaison idéale de techniques d'optimisation matérielle et logicielle pour obtenir des résultats supérieurs en utilisant un minimum de ressources informatiques sur de courtes périodes.

Pour notre modèle, l'entrée est la matrice dont chaque ligne représente un matériau et contient la liste triée des valeurs propres de sa matrice de Coulomb. On considère deux cas, le premier avec 5000 matériaux et le second avec tous les matériaux dans la base de donnée (47743 matériaux). On consacre 15 pourcent pour les tests et le reste pour l'entraînement.

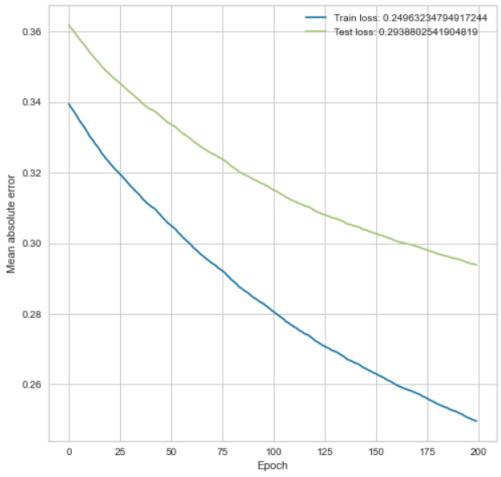
On retrouve les courbes du MAE et de prédiction en fonction de l'epoch dans les figures ci-dessus pour les deux cas

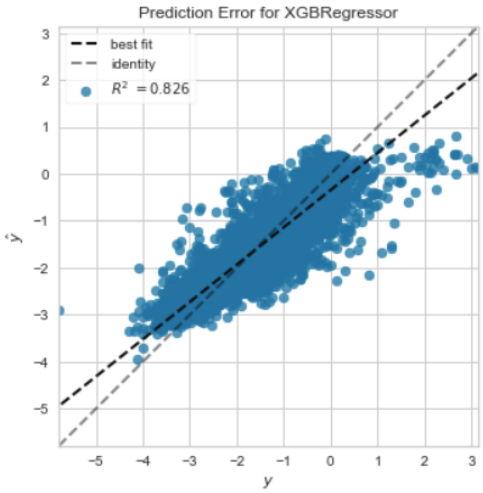





```
Validation mean absolute error:  0.3346640803670 86
```

Figure 23 – courbe du MAE et de prédiction pour le premier modèle






```
Validation mean absolute error:  0.28879650323643036
```

Figure 24 – courbe du MAE et de prédiction pour le deuxième modèle modèle

On remarque que l'erreur moyenne absolu est minimale pour le deuxième modèle ce qui est logique vu le nombre de donnée en entrée.

Remarque importante : On constate que le modèle de régression avec XGBoost est très performant et efficace en terme de durée d'entraînement.

CONCLUSION

En conclusion, ce modal a été l'occasion pour notre groupe de travailler pendant 3 mois sur une problématique réelle avec des données réelles. Il a également été l'occasion de découvrir de manière concrète la science des données.

Tout au long de ce travail, il a été question d'implémenter des algorithmes de traitement de données et de prédiction afin d'estimer l'énergie de formation d'un matériau. Deux grandes questions se sont naturellement posées : comment rendre nos données exploitables et comment les utiliser pour prédire l'énergie de formation d'un cristal.

Avant de répondre aux questions que nous nous étions posées, nous avons d'abord étudié les données, pour les comprendre et être en mesure de proposer des modèles efficaces. Une fois cette étape terminée, nous avons pu commencer à étudier la question de prédiction. A l'aide des compétences que nous avions acquises durant les premières semaines, nous avons réussi à manipuler les données pour les rendre exploitables, et pouvoir entraîner nos premiers modèles, en les adaptant aux spécificités des données. Nous avons utilisé principalement deux ensembles de données, un de taille 5304 rassemblant uniquement des cristaux contenant du carbone et un deuxième plus large de taille 47 743 et qui ne se restreint pas à un type particulier de cristaux mais limite uniquement le numéro atomique Z_{max} à 40.

Nous avons ensuite décidé de représenter nos données sous forme de matrices de listes suivant le descripteur utilisé : SOAP ou la liste triée des valeurs propres de la matrice de Coulomb. Le travail effectué à cette étape nous a permis de passer à la deuxième question, c'est-à-dire la prédiction.

En exploitant le travail effectué pour multiplier les descripteurs et la taille des données, nous avons réussi à prédire l'énergie avec des erreurs de l'ordre de 0.1 pour des énergies de formations variant dans l'intervalle $[-4, 4]$ avec le jeu de données le plus grand, ce qui semble être tout à fait correct.

Finalement, nous avons réussi à respecter la feuille de route que nous nous étions posée au début du projet et le cahier de charge est finalement rempli. Pourtant, il reste toujours des améliorations à faire en ce qui concerne les modèles de prédiction qu'on a utilisés en essayant d'optimiser encore plus le nombre de couches cachées pour chaque modèle par exemple. Un autre point d'amélioration est l'explicabilité de ces modèles en terme d'entrée convenable ; pourquoi par exemple avec le SOAP on obtient des résultats avec une perte très petite pour les matériaux contenant du carbone par rapport au deuxième jeu de données.

BIBLIOGRAPHIE

[1] YU, Haofeng., *Deep Blue to DeepMind : What AlphaGo Tells Us. Predictive Analytics and Futurism*, 2016, vol. 13, p. 42-45

[2] M. Ryan, K., Lengyel, J. Shatruk, *Crystal structure prediction via deep learning*, J. Am. Chem. Soc. 140, 10158–10168 (2018)

[3] Zheng, X., Zheng, P. Zhang, R.-Z., *Machine learning material properties from the periodic table using convolutional neural networks.*, Chem. Sci. 9, 8426–8432 (2018)