Utilisation de réseaux neuronaux pour la dynamique moléculaire.

Alexandre Antezak, Ewen Frogé
MU4PY115 machine learning

6th January, 2021

Abstract

La dynamique moléculaire est une méthode numérique permettant de décrire l'évolution temporelle d'un système de molécules. Des techniques dites *ab initio* reposant par exemple sur la théorie de la fonctionnelle de la densité, sont été utilisées. Ces méthodes permettent de connaître les surfaces d'énergie potentielle grâce aux principes de la mécanique quantique. Il s'agit de calculs coûteux en temps (millions d'heures de calcul). Les réseaux neuronaux sont le nouvel outil pressenti pour réduire ces temps de calcul. Il s'agit de gros algorithmes de régression assez mal théorisés mathématiquement mais ayant fait leurs preuves sur plusieurs problèmes d'informatique et de biologie. Nous allons au cours de ce projet utiliser cet outil pour prédire l'énergie de configuration d'une molécule de zundel à différents instants de temps.

1 Introduction

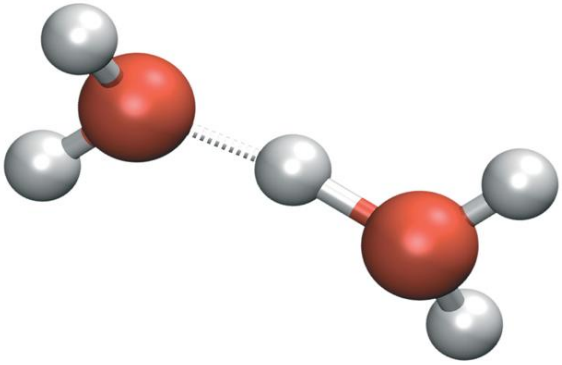
Les problèmes de dynamique moléculaire apparaissent au cours du milieu du vingtième siècle. On cherche à comprendre le comportement des liquides et on construit des modèles où les molécules du liquide sont considérées comme des sphères dures. Retrouver les trajectoires des molécules donnerait alors le comportement global du liquide. Dans les années 1970, on développe un outil mathématiques appelé théorie de la fonctionnelle de la densité (DFT). On utilise désormais la densité électronique et non plus les fonctions d'ondes individuelles pour caractériser un système quantique. On transforme un problème à N corps en un problème à 1 corps. En connaissant la densité électronique d'un système on le détermine totalement. Ce résultat est dû aux théorèmes de Hohenberg et Kohn. Ce qu'il faut retenir de cet outil est qu'il nous permet de calculer les valeurs moyennes d'observables en ne connaissant uniquement l'état fondamental du système[1]. Ainsi on dispose d'une surface d'énergie potentielle qui permet de prédire la dynamique du système. Résoudre les équations de la DFT recourt à de gros coûts en terme de calcul et limite donc les intervalles de temps de simulation d'évolution qui sont de l'ordre de la centaine de pico secondes.

On doit donc mettre en place de nouveaux outils. C'est là qu'interviennent les réseaux neuronaux. On utilisera le terme anglais neural network (NN). Il s'agit d'algorithmes de régression linéaire assez mal théorisés mathématiquement mais ayant fait leurs preuves dans les domaines de l'informatique et de la biologie. On discutera plus loin son fonctionnement. Ce travail s'appuie sur les résultats de l'article de Jörg Behler and Michele Parrinello[2], de celui de Pankaj Mehta[3] et a été supervisé par Julien Heu et Timothée Devergne, les responsables de l'UE.

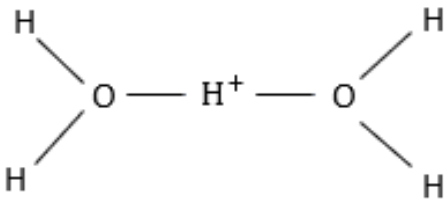
Au cours de ce projet nous allons utiliser un NN pour déterminer l'énergie de configuration d'une molécule de zundel. Il s'agit de deux molécules d'eau reliés par un proton (cf figures 1a et 1b).

Nous disposons des positions de chaque atomes de la molécule et l'énergie de configuration pour 0,5 ns avec un pas de temps de 0.5 fs à une température de 100K. Les énergies sont données en hartrees avec $1\,\mathrm{hartree} = 27,211396641308\,\mathrm{eV}$. Les positions sont données en Ångströms. Nous disposons au total de 999990 inputs d'énergie et de position.

Enfin ce réseau de neurones va nous permettre de réaliser des simulations dites de Monte-Carlo. Ces simulations sont utiles pour vérifier l'efficacité du réseau et aussi de se passer des simulations par DFT.



(a) représentation des atomes du zundel.[4]



(b) formule développée plane du zundel.[5]

Fig. 1. molécule de zundel.

2 Outils et méthodes

2.1 NN

2.1.1 Définition d'un NN

Afin de prédire les énergies de configuration de la molécule de zundel, nous allons utiliser un NN. Comme expliqué brièvement précédemment, cet algorithme de régression linéaire est de plus en plus utilisé dans la communauté scientifique. L'utilisation d'un NN présente plusieurs étapes. Tout d'abord sa construction. Un NN se décompose sous la forme de couches, la première étant la couche d'inputs où l'on rentre les données à traiter, et la dernière celle d'output qui donne les prédictions attendues. Entre ces deux couches se trouvent les couches cachées. Ce sont ces couches qui permettent d'effectuer les calculs de régression. Il existe plusieurs types de NN, dans notre projet nous utilisons un NN classique dit feddforward dans lequel tous les noeuds du réseau sont reliés ensembles (cf figure 2). A chaque lien est associé un poids qui est un paramètre à optimiser. La seconde étape est celle de l'entraînement. On utilise un set d'entraînement qui va nous permettre d'ajuster les poids du NN afin de prédire un output similaire au résultat connu. Cette optimisation est possible grâce à des algorithmes

de descente de gradient stochastique. Enfin la dernière étape est celle du test. On entre dans le NN un set de test et on compare les outputs du NN avec les vrais résultats.

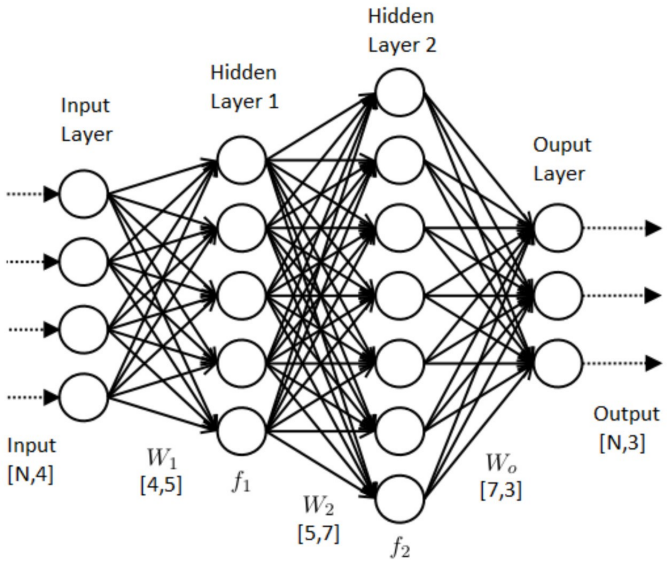


Fig. 2. Exemple d'un NN[6].

2.1.2 Construction du NN pour le zundel

Le zundel est une molécule composée de 5 hydrogènes et 2 oxygène. L'hypothèse que nous faisons est que chaque atome porte une partie de l'énergie totale de la molécule. Cette hypothèse est discutable sur la plan physique mais nous permet de construire notre NN. Ainsi nous allons construire, grâce au package keras[7], un NN qui sera composé de 7 subnets. Un subnet est un réseau modèle et un input. On aura ainsi 2 réseaux modèles, un pour chaque élément à savoir oxygène et hydrogène (cf figure 3). On ne peut pas considérer ces 7 subnets indépendamment car la grandeur physique dont nous disposons est l'énergie de configuration de la molécule et pas l'énergie par atome. Nous devons désormais construire les réseaux modèles. En inputs on injectera ce que l'on appelle un descripteur et sera détaillé dans la section suivante. Puis nous ajoutons 2 couches cachées de 30 noeuds tous reliés entre eux comme dans la figure 2,ce sont des couches denses. On utilisera une tangente hyperbolique comme fonction d'activation et Adam comme optimizer. L'optimizer est la fonction qui va permettre d'effectuer la descente de gradient pour ajuster les poids du réseau. On choisira la mean squared error comme loss, présentée par l'équation 2. Ces choix sont justifiés par l'expérience. Enfin la couche d'output correspond à "l'énergie de l'élément considéré".La somme des ces outputs correspond à l'output du NN qui est l'énergie totale de la molécule.

2.2 Descripteurs

Discutons à présent des inputs du NN. Nous disposons des postions de chaque atome. Nous ne pouvons pas utiliser ces postions en input pour plusieurs raisons. Premièrement les

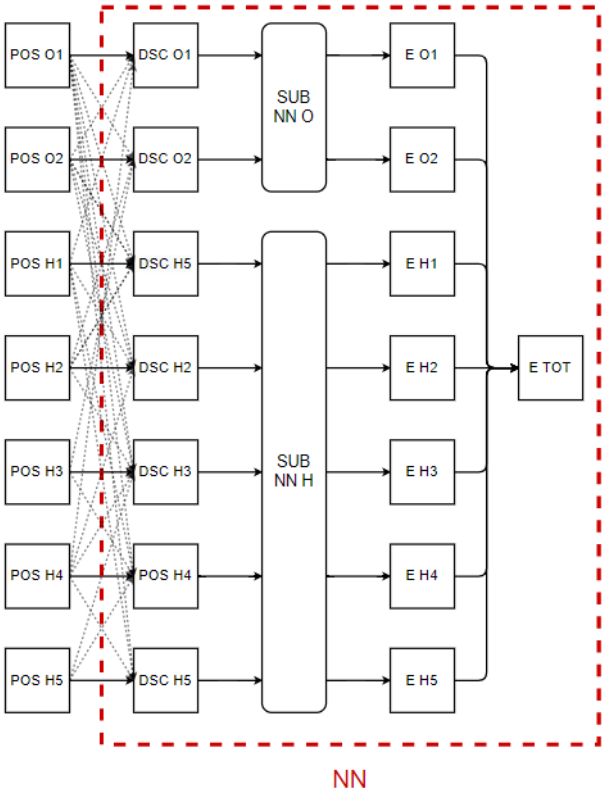


Fig. 3. NN du Zundel.

inputs doivent être invariants par symétrie. En effet l'énergie de la molécule est la même si on la translate ou la tourne d'un certain angle. C'est donc un point qu'il faut "préciser" au NN et dont il ne doit pas dépendre. Deuxièmement les positions des atomes ne sont pas décorrélées, elles sont dépendantes l'une de l'autre à cause des répulsions des nuages électroniques. Nous devons donc utiliser un outil prenant en compte les interactions entre voisinage électronique de chaque élément. Cet outil est ce que l'on appelle descripteur. Ce descripteur est obtenu grâce à l'outil physique Smooth Overlap Atomic Positions (SOAP). Comme expliqué dans la documentation du package Python DScribe[8], dont nous nous sommes servis pour nos simulations, "SOAP est un descripteur qui encode des régions de géométries atomiques en utilisant une expansion locale d'une gaussienne étalée de la densité atomique." On obtient en sortie de la fonction SOAP le partial power spectrum défini par un vecteur **p(r)** dont les composantes se calculent de la manière suivante

$$p(\mathbf{r})_{nn'l}^{Z_1Z_2} = \pi\sqrt{\frac{8}{2l+1}}\sum_m c_{nlm}^{Z_1}(\mathbf{r})^* c_{n'lm}^{Z_2}(\mathbf{r}), \tag{1}$$

où $Z_1, Z_2 = \mathrm{O}, \mathrm{H}$ et n, n', l, m sont les nombres quantiques et $\mathbf{r}$ le vecteur position. Les coefficients $c_{nlm}^{Z_1}(\mathbf{r})$ sont obtenus en décomposant la gaussienne lissée de la densité atomique de chaque atome sur la base des harmoniques sphériques et des fonctions radiales.

Pour calculer ces descripteurs nous devons préciser plusieurs paramètres: $\mathrm{n_{max}}$ et $\mathrm{l_{max}}$ les nombres quantiques maximaux, $\mathrm{r_{cut}}$ le rayon de coupure au delà duquel la densité atomique est nulle. La largeur de la gaussienne $\mathrm{sigma_{soap}}$ doit aussi être fixée. Une fois ces paramètres fixer (ils ne sont pas encore optimisés), nous disposons d'un vecteur contenant les descripteurs de chaque atome que nous pouvons entrer comme input dans le NN.

2.3 Principal Component Analysis (PCA)

La PCA est un outil permettant de réduire la dimensionnalité d'un ensemble de point dans un hyper espace. L'idée est de transformer les variables corrélées dans cet espace en des variables décorrélées les unes des autres (cf figure 4). On va observer des directions de variance dominante et d'autres de variance négligeable par rapport à celles dominantes. Ainsi on va pouvoir mettre à jour des directions privilégiées concentrant un ratio de variance important. Cette méthode permet de réduire le nombre d'inputs et faciliter le travail du NN et sa rapidité d'exécution.

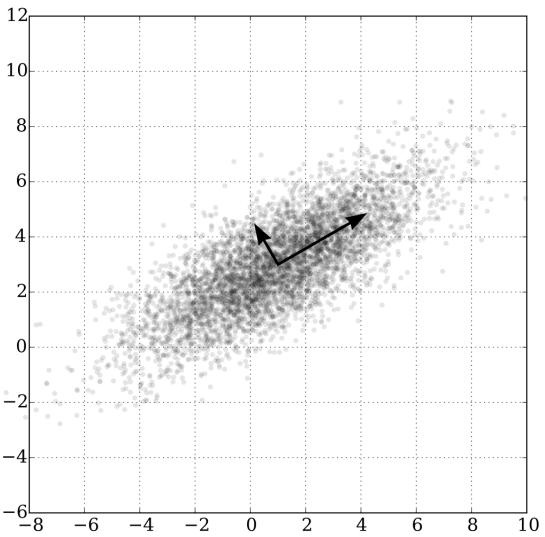


Fig. 4. Exemple d'une PCA[9].

2.4 Echelonnage des inputs

Avant toute simulation il est important d'échelonner (scale) les données notamment lorsque nous faisons une PCA. Sans ce scaling on pourrait se retrouver avec des nouvelles variables post PCA qui n'auraient pas lieu d'être. On peut penser à un cercle qui est transformé en ellipse si on multiplie par un coefficient l'un des axes du graphique. Pour effectuer notre scaling nous disposons grâce au module preprocessing de sklearn de 3 scalers[10]: Standard scaler, Max Abs Scaler et Min Max scaler. Le premier permet de transformer le set de données en un set de moyenne nulle et de variance unité. Il est utile lorsque la distribution initiale est de

type gaussienne. C'est ce que nous observons pour les énergies (cf figure 5). Le second est défini dans la documentation de sklearn comme "un estimateur qui scale et translate chaque composante individuellement de sorte que la valeur absolue maximale de chaque composante dans le set d'entraînement soit 1.0 ." Enfin le dernier scaler effectue la même opération sur la valeur maximale et scale et translate chaque composante de sorte que la valeur minimale soit 0.

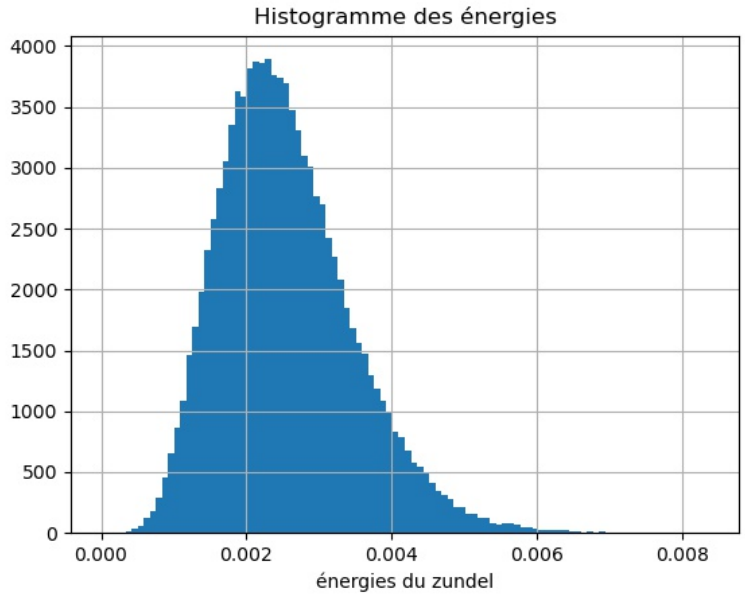


Fig. 5. Histogramme des énergies pour la molécule de zundel.

Nous avons tout d'abord réalisé un scaling par atome, ce qui a permis d'atteindre une précision très élevée. Cela dit, ce scaling pose un problème physique puisque l'on ne considère plus les hydrogènes comme équivalents. Il faut donc considérer un scaling par élément. Nous avons utilisé le Standard scaler par expérimentation. Ce travail de scaling permet d'améliorer les prédictions du NN car il travail avec des données dans des ordres de grandeur plus proches qu'avant le traitement. Une fois le NN entraîné et les énergies prédites on utilise la méthode inverse transform permettant de plot les vraies valeurs de l'énergie et non pas celles scalées. On applique cette méthode sur les énergies prédites par le NN.

2.5 Optimisation

On cherche à trouver les paramètres extérieurs à notre NN, dits hyperparamètres, tel que celui ci fonctionne de manière optimale. Pour cela, on cherche dans un espace d'hyperparamètres le point pour lequel la loss est minimale. La loss est une fonction qui mesure l'écart entre les valeurs prédites par notre NN les vraies valeurs attendues. Ici on a choisi d'utiliser la Mean Squared Error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}[E_i - \hat{E}_i]^2, \tag{2}$$

avec E_i l'énergie prédite et $\hat{E}_i$ l'énergie obtenue par DFT.

Pour des fonctions classiques , une recherche de minimum se fait à l'aide des dérivées, dans un espace uniforme. Ici cependant, notre fonction est très complexe, et faire une recherche pareille serait extrêmement coûteuse en temps. Rien que pour obtenir la loss correspondante à un set de paramètre, il faut entraîner le NN entièrement. Nous avons donc procédé par recherche Bayesienne: nous donnons pour chaque hyperparamètre un ensemble discret de valeurs possibles, nous cherchons parmi celles ci laquelle permet de minimiser notre loss. Nous avons donc commencé par faire une recherche Bayesienne générale, sur un ensemble d'hyperparamètres: cela permet de prendre en compte les corrélations entre les différents paramètres et d'avoir une idée générale des ordres de grandeur que devrait avoir chaque hyperparamètre. C'est cependant très long, et l'espace tout entier ne peut généralement pas être exploré car la durée du processus scale de manière exponentielle avec le nombre d'hyperparamètres. Nous complétons donc cette première évaluation avec une recherche Bayesienne paramètre par paramètre, qui nous permet de valider les résultats obtenus par la première évaluation ainsi que d'avoir des valeurs plus précises, car ayant un espace d'une seule dimension, on peut se permettre de donner plus de valeurs possibles parmi lesquelles explorer.

2.6 Monte-Carlo

Pour mettre à profit notre NN et étudier la dynamique de la molécule, nous utilisons un procédé stochastique appelé méthode de Monte-Carlo. Ces algorithmes sont des algorithmes aléatoires visant à obtenir un résultat approché de la valeur attendue. Dans notre cas, cela permet de remplacer la dynamique (vitesses/forces) par un processus aléatoire utilisant uniquement les positions et énergies. Pour cela, on va modifier la position des particules de manière aléatoire depuis une position initiale. Ce déplacement est limité à une boite de coté delta, que l'on déterminera en fonction de ce que l'on appelle taux d'acceptation. Il correspond à la probabilité de l'acceptation ou du refus d'une position. Ensuite, grâce a notre NN, on estime l'énergie de cette nouvelle configuration: si celle ci est plus petite que notre énergie de départ, on l'accepte comme configuration possible au temps suivant. On autorise aussi la différence d'énergie acceptée à être très faiblement positive, car dans le cas contraire, notre système pourrait se retrouver coincé dans un petit puits de potentiel alors qu'en dynamique, il ne le serait pas grâce aux moments que sont la vitesse et l'accélération. Si la différence est plus grande que le seuil fixé (qui dépend de delta), on refuse le déplacement et on reste donc dans la même configuration au temps suivant. On peux alors répéter l'opération pour le 2ème pas de temps, et ainsi de suite de manière récurrente.

On peux ensuite comparer les prévisions de l'évolution du système par DFT et celle obtenu en utilisant notre NN en opposant leur fonctions de distribution radiale g(r). Cette fonction permet de décrire la variation de la densité de présence d'une paire en fonction de la distance depuis une particule.

3 Résultats

3.1 Prédiction des énergies

La phase de mise ne place du NN est suivie par une phase d'optimisation des paramètres. Pour cela nous avons utilisé hyperopt qui est un package[11] permettant de minimiser une fonction dépendant de plusieurs paramètres. Dans notre cas la fonction à minimiser est une loss MSE. Les paramètres à optimiser sont: $n_{\max}, l_{\max}, r_{\text{cut}}, \text{sigma}_{\text{soap}}$, les scalers utilisés et le nombre de couches du modele, ainsi que le nombre de noeud de la couche. Après avoir laissé exécuté l'optimisation, nous avons nos paramètres optimisés:$n_{\max} = 4, l_{\max} = 5, r_{\text{cut}} = 11.0, \text{sigma}_{\text{soap}} = 1$. On trouve également que le nombre de couches pour le modèle est 2 avec une densité 30, et que le scaler utilisé devrait être un Standard-Scaler, qui retranche la moyenne aux données et les divise par l'écart type . On a observé que le r_{cut} n'est pas vraiment optimisable puisque nous obtenons de meilleurs résultats en augmentant cette grandeur. Cependant le r_{cut} est un outil permettant de considérer des phénomènes locaux et donc s'il est trop grand, il n'a plus de sens physique dans notre problème. La PCA nous permet de réduire la dimension des descripteurs de 216 à 93 en prenant un ratio de variance de 99.9999%. Cependant ce pourcentage est choisit arbitrairement en essayant de maximiser la rapidité du NN et minimiser la loss. A partir d'un certain pourcentage (ici 99.9999% fonctionne très bien) la minimisation de la loss devient négligeable et augmenter ce ratio ne fait que ralentir l'entraînement du réseau. Alors pourquoi faire une PCA? Et bien en sortie de PCA les nouvelles variables sont triées par ordre d'importance dans le ratio de variance et cela a pour effet (expérimental) d'accélérer l'entraînement.

La phase d'entraînement du réseau est réalisé avec une batch size de 10 et un slicing des données de 5 (nous n'avons pas considérer toutes les données car la ram du pc n'était pas assez importante). Nous avons mis en place deux callbacks, le premier appelé early stoping permettant d'arrêter l'entraînement si la loss ne s'améliore pas sur un nombre prédéfini d'epoch, c'est ce qu'on appelle patience. Le second callback est le learning rate reducer permettant de multiplier le learning rate d'un facteur prédéfini si la loss ne s'améliore pas sur un nombre prédéfini d'epoch aussi appelé patience. Dans notre cas nous avons une patience de 20 pour le early stopping et une patience 4 avec un facteur 0.5 pour le learning rate reducer. Nous avons entraîné le NN sur un set d'entraînement représentant 85% du set total. Finalement après environ 3 heures d'entraînement sur 258 epochs, en traitant une epoch en environ 1 min 30 et en diminuant le learning rate jusqu'à 1.2207×10^{-7}, nous obtenons une loss de $0,0005$ et ce graphique de la loss du set d'entraînement et de validation en fonction des epochs (cf figure 7). La loss du set de validation est bruitée car nous entraînons le réseau sur un très petit batch size. Nous pouvons désormais utiliser notre réseau pour calculer les énergies du set de test. Ce set de test représente 10% du set total. Nous faisons un inverse-scaling des énergies obtenues pour travailler avec une grandeur physique et traçons le graphique des énergies prédites en

fonction des énergies obtenues par DFT (cf figure 6). Nous trouvons une erreur moyenne de 0.4 meV.

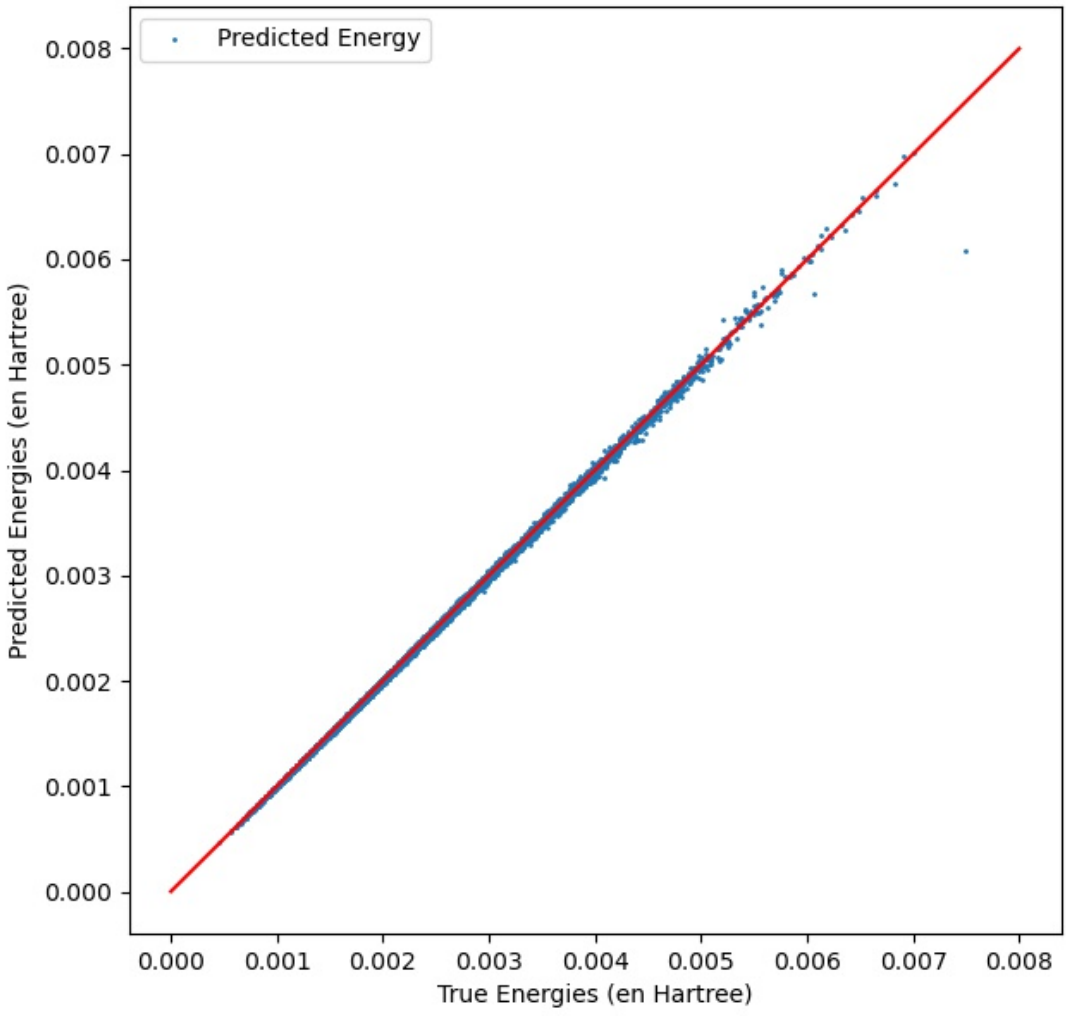


Fig. 6. Énergies du set de test prédites par le NN en fonction des énergies obtenues par DFT.

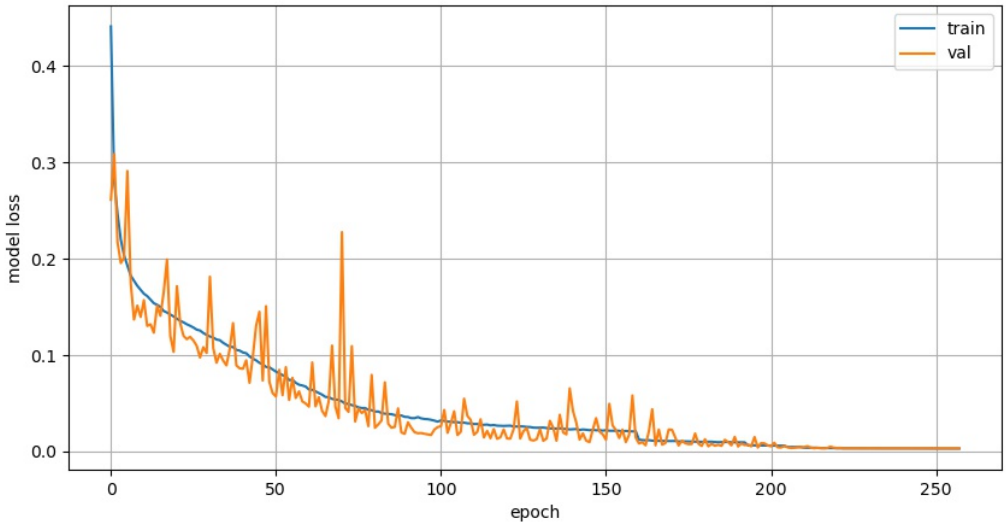
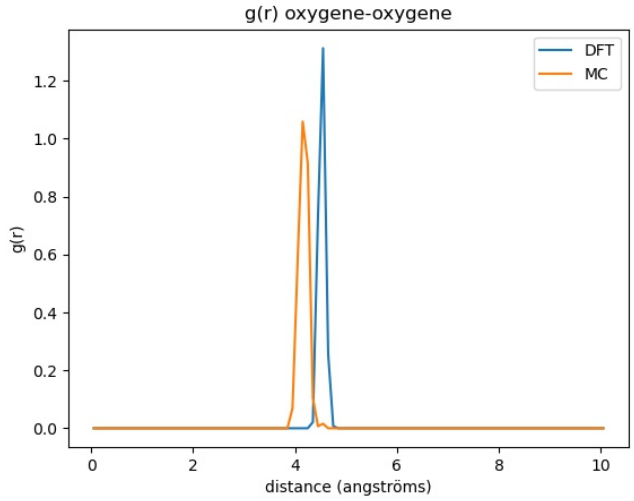


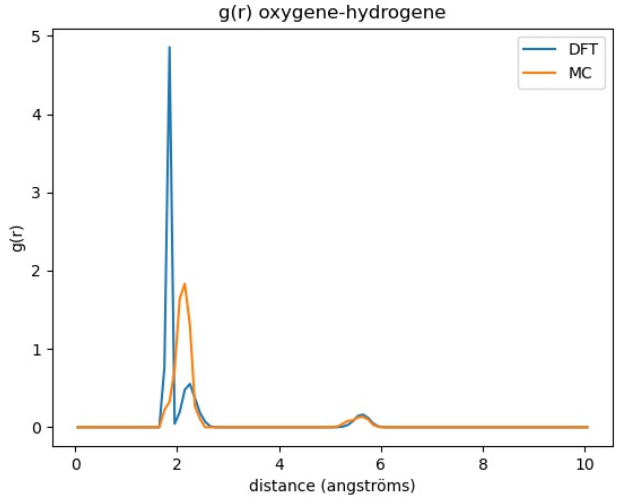
Fig. 7. Loss du NN sur le set d'entraînement et de validation en fonction de l'epoch.

3.2 Simulations de Monte-Carlo

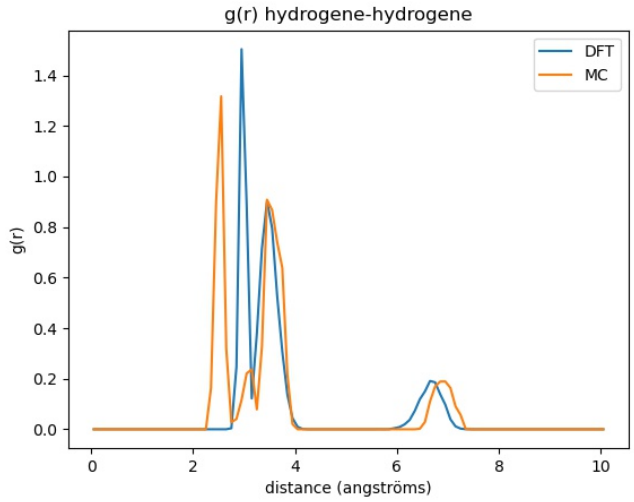
Le réseau étant optimisé et entraîné, nous pouvons l'utiliser pour prédire des énergies correspondantes à 20 000 positions données. Avec la méthode présentée dans la section 2.6, nous avons calculé les fonctions radiales des paires oxygène-oxygène, oxygène-hydrogène et hydrogène-hydrogène. Ces calculs ont été faits sur le logiciel VMD. Voici les graphiques que nous obtenons.



(a) g(r) oxygène-oxygène Monte-Carlo et DFT.



(b) g(r) oxygène-hydrogène Monte-Carlo et DFT.



(c) g(r) hydrogène-hydrogène Monte-Carlo et DFT.

Fig. 8. g(r) MC/DFT pour les différentes paires.

4 Discussion

Les résultats que nous avons obtenus sont très satisfaisants. Il y a cependant des pistes que nous n'avons pas explorées qui pourraient améliorer nos résultats. Premièrement nous considérons les cinq hydrogènes équivalents ce qui physiquement n'est pas vrai puisque l'un d'entre eux est un proton. Ce proton peut être échangé avec un hydrogène de la molécule. A basse température ce phénomène arrive très rarement et nous ne l'avons pas pris en compte dans notre travail puisque nous travaillons à 100K. Avec des datasets à plus haute température il aurait fallu effectué un travail de classification pour déterminer les échanges proton-hydrogènes. Ces datasets auraient été intéressants à utiliser pour entraîner notre NN mais nous n'avons pas pu les utiliser par manque de temps.

5 Conclusion

Nous avons créé un outil, un réseau neuronal qui est un important algorithme de régression linéaire, permettant de prédire une énergie de configuration d'une molécule de zundel en fonction de la position de chacun de ses atomes. Grâce à divers outils, nous avons optimisé ces prédictions et sommes arrivés à déterminer les énergies de configuration avec une erreur moyenne de 0.4 meV (cf figure 6). A titre de comparaison les erreurs avec la méthode DFT sont de 1 à 2 meV/atoms. Ainsi ces erreurs rentrent dans les barres d'erreurs des calculs par DFT. Ce réseau nous a permis d'effectuer des simulations de Monte-Carlo et de montrer qu'elles peuvent remplacer les simulations par DFT très coûteuses en temps de simulation (cf figure 8).

6 References

1. Wikipedia. *Théorie de la fonctionnelle de la densité* https://fr.wikipedia.org/wiki/Th%C3%A9orie_de_la_fonctionnelle_de_la_densit%C3%A9 (2021).
2. Behler, J. & Parrinello, M. Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces. *Physical Review Letters* (2007).
3. Mehta, P. A high-bias, low-variance introduction to Machine Learning for physicists. *arXiv:1803.08823v3 [physics.comp-ph] 27 May 2019* (2019).
4. Wikipedia. *Hydronium* https://en.wikipedia.org/wiki/Hydronium (2021).
5. *formule plane du Zundel* https://github.com/HeuJulien/ML-P6-2020/blob/master/projects/zundel.png (2021).
6. Ahire, J. B. *The Artificial Neural Networks handbook: Part 1* https://medium.com/coinmonks/the-artificial-neural-networks-handbook-part-1-f9ceb0e376b4 (2021).
7. *Keras* https://www.tensorflow.org/api_docs/python/tf/keras (2021).
8. Himanen, L. *DScribe Documentation* https://singroup.github.io/dscribe/latest/about.html (2021).
9. Wikipedia. *Analyse en composantes principales* https://fr.wikipedia.org/wiki/Analyse_en_composantes_principales (2021).
10. *Sklearn Documentation* https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing (2021).
11. *Hyperopt Documentation* http://hyperopt.github.io/hyperopt/ (2021).