Problèmes spectraux en mécanique quantique avec réseaux de neurones - rapport de stage

Encadrants : Jérôme Margueron et Hubert Hansen
Étudiant : Clément Lotteau

Mai - Juin 2020

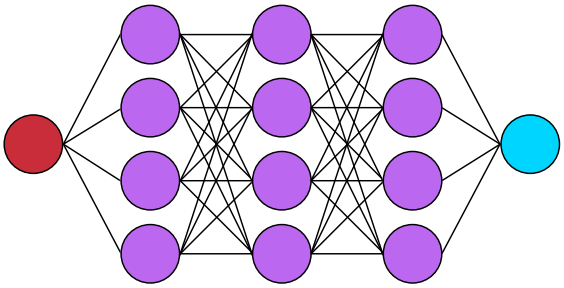

FIGURE 1: Schéma d'un réseau de neurones.

Résumé

Certains systèmes quantiques ne peuvent être étudiés analytiquement par la résolution de l'équation de Schrödinger indépendante du temps. Pour cette raison, les méthodes numériques permettant de résoudre cette équation sont centrales en physique moderne car elles nous permettent d'étudier des potentiels réels. Ce rapport de stage présente les résultats de deux méthodes de résolution développées en autonomie : un algorithme de diffusion Runge-Kutta d'ordre 4, et une approche stochastique originale des réseaux de neurones s'inscrivant dans le cadre des méthodes variationnelles et reposant sur le théorème d'approximation universelle [1].

.

Table des matières

Introduction

Le but de ce stage était de calculer numériquement la fonction d'onde et l'énergie d'une particule piégée dans un potentiel quelconque. Il s'agit un problème général en mécanique quantique s'étendant de la physique atomique à l'étude des quarks. Pour ce faire, j'ai exploré deux méthodes : un algorithme Runge-Kutta d'ordre 4 permettant de trouver les états liés et leurs énergies, et une méthode de minimisation stochastique de l'énergie utilisant un réseau de neurones permettant de trouver l'état fondamental. Dans les deux cas, on cherche à résoudre l'équation de Schrödinger indépendante du temps [2] :

$$\left[\frac{-\hbar^2}{2m}\frac{d^2}{dx^2} + V(x)\right]\psi(x) = E\psi(x) \tag{1}$$

Pour trouver l'énergie de $\psi(x)$, on multiplie (1) à gauche par $\psi(x)$ et on intègre l'énergie cinétique par partie. Dans le cas particulier où $V(x) = \frac{1}{2}m\omega x^2$, on obtient :

$$E = \frac{\frac{-\hbar^2}{2m}\left(\left[\psi\psi'\right]_a^b - \int_a^b |\psi'|^2 dx\right) + \frac{1}{2}m\omega^2 \int_a^b x^2|\psi|^2 dx}{\int_a^b |\psi|^2 dx} \tag{2}$$

On obtient les solutions analytiques en intégrant de $-\infty$ à $+\infty$. On prend m, $\hbar$ et ω égaux à 1 par la suite :

$$E_c = \frac{\hbar\omega}{4} \quad ; \quad E_p = \frac{\hbar\omega}{4} \quad ; \quad E_{totale} = \frac{\hbar\omega}{2} \tag{3}$$

1 Méthode Runge-Kutta

État	Énergie
0	0.487663
1	1.46298
2	2.43830
3	3.41361
4	4.38893

TABLE 1: Énergies trouvées grâce à la propagation RK4

L'algorithme RK d'ordre 4 que j'ai programmé nous permet de trouver les énergies (table 1) et les fonctions d'ondes (figure 2) des états liés. On observe que, contrairement à la théorie, les énergies trouvées ne sont pas 0.5, 1.5, 2.5...etc. Une erreur importante est commise. Ce problème est peut-être dus à l'accumulation d'erreurs de calcul au fil de la propagation. L'écart entre chaque niveau est constant et vaut environ 0.97532 (contre 1 théoriquement). Des informations et des résultats supplémentaires sont disponibles sur mon GitHub [8].

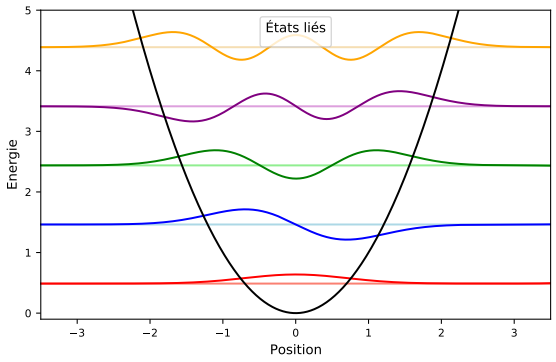

FIGURE 2: Tracer des états liés, les fonctions d'onde (non normalisées) sont alignées sur leurs énergies.

2 Réseaux de neurones

2.1 Concept et vocabulaire

Prenons un exemple fictif dans lequel on cherche à ce que notre réseau reproduise la fonction discrète suivante :

$$f(x) = \begin{cases} 1 & \text{quand} \quad x = 0 \\ 7 & \text{quand} \quad x = 1 \\ 3 & \text{quand} \quad x = 2 \end{cases} \tag{4}$$

On représente cette fonction par deux listes de trois nombres : $x = [0, 1, 2]$ et $y = [1, 7, 3]$ puis on entraine notre réseau à reproduire y lorsqu'on lui demande de faire une prédiction à partir de x. Pour ce faire, on commence par initialiser le réseau avec des paramètres aléatoires puis on lui fournit x et y.

Entrainement :

Epoch 1 : Le réseau fait une première prédiction $z = [32, -9, 64]$ éloignée de la liste attendue. La

fonction de coût calcule ensuite le *loss*, l'écart quadratique moyen entre y et z. On calcule ensuite le gradient de la fonction de coût par rapport aux paramètres du réseau pour savoir comment les modifier afin de minimiser le loss. On modifie les paramètres en conséquence, fin du premier epoch.

Epoch 2 : Le réseau prédit $z = [11, 3, 9]$. La prédiction est meilleure grâce au premier entrainement mais doit être améliorée. On calcule le loss, le gradient, on modifie les paramètres du réseau et on recommence le processus pour l'epoch 3. Ces calculs sont gérés automatiquement par Keras [7], la librairie Python que nous utilisons ici.

Géométries :

Les réseaux étudiés ici prennent une discrétisation de l'espace (x) en entrée et renvoient une fonction d'onde (z) en sortie. Entre les deux, les données sont propagées à travers des "couches cachées" (en violet sur la figure 1) dont on choisit le nombre de neurones. Par la suite, j'utiliserai le mot "couche" pour désigner les couches cachées des réseaux. Aussi, un réseau 200x200 par exemple désignera un réseau à deux couches cachées, toutes deux composées de 200 neurones.

```
#INITIALISATION DU RÉSEAU
model = models.Sequential([
  layers.Dense(100, input_shape=(1,), activation='relu'),
  layers.Dense(100, input_shape=(1,), activation='relu'),
  layers.Dense(1),
])
opt = optimizers.Adam(learning_rate=0.001)
model.compile(loss='mse',optimizer=opt)
#ENTRAINEMENT SUR 50 EPOCHS
model.fit(x,y,epochs=50,batch_size=50)
#PRÉDICTION APRÈS ENTRAINEMENT
predictions = model.predict(x)
```

FIGURE 3: Définition d'un réseau à deux couches cachées comportant chacune 100 neurones.

Des tutoriels d'initiation aux réseaux de neurones écrits par Colin Bernet sont disponibles en bibliographie : [3], [4] et [5]

2.2 Caractéristiques de la machine utilisée

Tout le travail lié aux réseaux de neurones a été réalisé sur Google Colab [6] qui permet d'exécuter des notebooks Python depuis un navigateur. Les caractéristiques de la machine sont : 2 processeurs Intel(R) Xeon(R) CPU @ 2.20GHz ; 1 coeur ; 46 bits physiques ; 48 bits virtuels

2.3 Fit d'une gaussienne

On cherche maintenant à reproduire l'état fondamental d'un oscillateur harmonique à une dimension. On dispose d'une liste x discrétisée de -5 à 5 sur 10001 points, et d'une liste y calculée à partir de la solution analytique de l'équation (1) page 1 pour l'état fondamental [2] :

$$\psi_{EF}(x) = \left(\frac{m\omega}{\pi\hbar}\right)^{\frac{1}{4}} e^{\frac{-m\omega x^2}{2\hbar}} \tag{5}$$

Les courbes noire, rouge et orange de la figure 4 illustrent une fonctionnalité de Keras permettant de mesurer le loss à chaque epoch. La courbe bleue est une moyenne sur 30 runs du programme de reproduction. On voit que sur la courbe rouge que le minimum de la fonction de coût a été atteint vers le 45^e epoch environ. Néanmoins, le réseau continue à modifier ses paramètres lors des epochs suivants et oscille autour du minimum de la fonction de coût. Le loss augmente et les prédictions s'éloignent de la cible. L'aléatoire devient important à partir du 30^e epoch environ. Autrement dit, au delà de ce seuil, deux runs du programme pourront donner des résultats très différents.

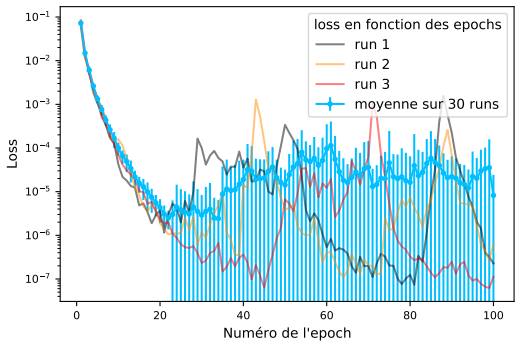

FIGURE 4: Réseau : 200x200x200 neurones.

Est-ce que je mentionne ça alors que je n'ai pas la place de détailler les résultats ? Une étude du loss en fonction des epochs a été faite pour comparer l'impact de la géométrie sur la minimisation du loss [8]. On y voit que l'ajout de non linéarité (plusieurs couches) permet d'atteindre un loss plus faible, et ce en moins d'epochs que pour un réseau à une couche. On y voit aussi que l'augmentation du nombre de paramètres permet d'obtenir des loss plus bas mais que les oscillations autour du minimum de la fonction de coût ont une plus grande amplitude qu'avec des réseaux moins denses.

REFAIRE LES ÉTUDES AVEC PLUSIEURS GÉOMÉTRIES.

2.4 Calcul de l'énergie d'une prédiction du réseau

Une fois l'état fondamental reproduit par le réseau, on peut en extraire l'énergie avec l'équation (2) page 1. Une première tentative d'extraction nous avait posé problème du fait que les prédictions du réseau sont des morceaux de droites. Un bon apprentissage permet de lisser le résultat mais la dérivée de la prédiction restait discontinue. Nous avons donc extrait l'énergie à partir de splines cubiques de la prédiction dont le calcul [10], la dérivation et l'intégration [9] sont gérés par la librairie SciPy. La figure 5 page 3 présente les résultats du calcul des énergies et de la norme de 50 prédictions effectuées par le réseau après entrainement. On observe que l'énergie totale des prédictions est toujours supérieure à l'énergie théorique de 0.5, celle-ci étant le minimum atteignable, ce qui illustre le fait qu'on se place dans le cadre des méthodes variationnelles.

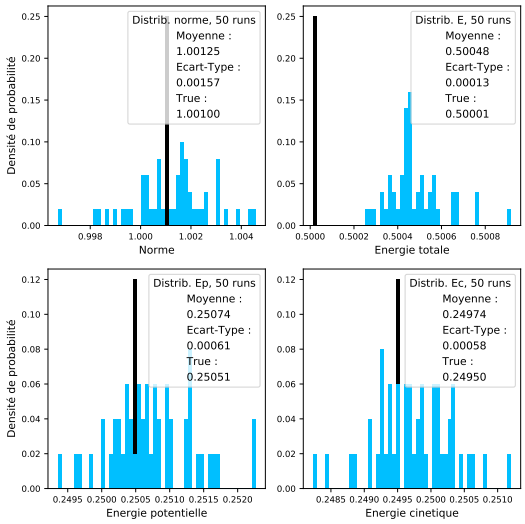

FIGURE 5: Les barres noires sont des repères visuels de la fonction à fitter.

3 Minimisation de l'énergie : utilisation stochastique des réseaux de neurones

Jusqu'ici, nous nous sommes servis de la solution analytique pour entrainer le réseau puis nous avons extrait l'énergie de ses prédictions. Notre but maintenant est de trouver une méthode pour que le programme retrouve l'énergie et la fonction d'onde de l'état fondamental sans connaître la solution au préalable. Pour résoudre ce problème, j'ai eu l'idée du programme détaillé en figure 7 page 4 dans lequel le réseau cherche à reproduire une fonction d'onde cible et où on se sert des petites fluctuations (figure 4 page 3) pour nous rapprocher de l'état d'énergie minimale.

3.1 Première version du programme

Dans la première version du programme, la première cible était une fonction composée de nombres aléatoires entre 0 et 1 et l'étape 6 n'existait pas. De plus, je détruisais le réseau puis le reconstruisais à l'étape 8 en espérant générer un grand loss et ainsi accélérer le processus de minimisation. Cette méthode a donné le résultat visible sur la figure 6.

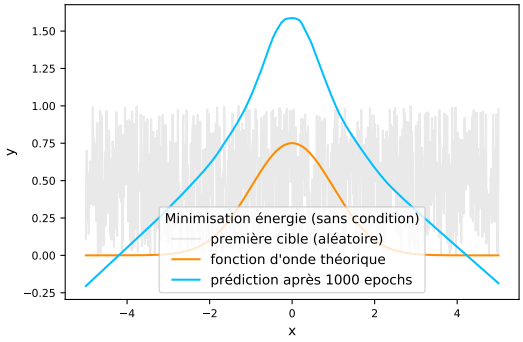

FIGURE 6: Première version du programme. 1000 itérations, discrétisation sur 1000 points.

On voit que les prédictions tendaient vers une gaussienne au bout de 1000 itérations mais le processus était particulièrement lent et les résultats trop peu précis. J'ai donc ajouté 3 conditions (figure 7, étape 6). Dorénavant, la fonction d'onde doit être : positive, symétrique et normée. Les deux premières conditions donnaient de meilleurs résultats que la figure 6 mais la précision restait faible. C'est la normalisation qui a permis d'atteindre des résultats similaires à la figure 8 page 5. La déconstruction-reconstrution du réseau avec des paramètres aléatoires avait permis d'obtenir les résultats de la figure 6, mais elle faisait perdre beaucoup de temps au programme lorsque les prédictions était déjà bonnes. Il ne restait plus qu'à affiner le résultat mais le programme continuait de reprendre son apprentissage à 0 à chaque fois qu'une meilleure cible était trouvée. Autrement dit, si on regarde la figure 4 page 3, on voit que le loss est grand au début de l'entrainement. Cela signifie que les prédictions du réseau sont très éloignées de la cible alors qu'on cherche de petites variations pour affiner le résultat. De plus, les nouvelles conditions permettaient déjà d'orienter le programme vers une gaussienne.

3.2 Dernière version du programme

Dans cette version, j'ai abandonné la décons.-recons. du réseau et j'ai aussi changé la discrétisation en passant de 1000 à 100 points pour diminuer le temps de calcul. La figure 7 présente la dernière version en date du programme de minimisation dont le code commenté en anglais est disponible sur mon GitHub [8].

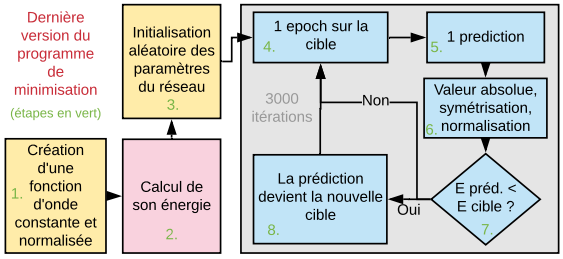

FIGURE 7: La première cible est une fonction constante. On garde la prédiction avec l'énergie la plus faible en sortie de boucle.

Le paragraphe suivant devrait peut-être aller dans la vidéo ?

Un parallèle peut être fait entre le programme et la méthode du recuit simulé. Chaque itération du programme "réchauffe" le système (le réseau) via un epoch et ses paramètres changent légèrement. Néanmoins, à la différence du recuit, la sélection des prédictions nous permet de trouver la meilleure configuration des paramètres en vue de minimiser l'énergie du système quantique étudié et non d'un paramètre du réseau. On peut ainsi voir ce programme comme un hybride mélangeant le recuit et les réseaux de neurones.

Pour étudier la convergence du programme, on calcule l'écart entre l'énergie trouvée et l'énergie

théorique (équation (3) page 1) :

$$E_{programme} - 0.5 \tag{6}$$

On trace ensuite cet écart en fonction du temps de calcul CPU pour observer la convergence. Le résultat est illustré par la figure 13 page 7 sur laquelle on a fait la moyenne des écarts et du temps CPU sur 30 runs du programme. On y voit notamment que la convergence dépend beaucoup du nombre de paramètres du réseau et de la géométrie choisie. Par exemple, les réseaux à une couche et les réseaux à 500 paramètres ne convergent pas. On remarque aussi que, pour les réseaux à 6 couches, 5000 paramètres semble être un meilleur choix que 30600, contrairement aux réseaux à 3 couches.

La figure 8 illustre un résultat moyen de l'algorithme après 2000 itérations. On peut y voir que l'écart entre la fonction d'onde théorique et le résultat de l'algorithme semble être le plus important là où la courbure est forte. On voit que la partie linéaire est inférieure à la théorie alors que le sommet est supérieur. D'autres résultats semblaient comporter les mêmes problèmes. Le but du réseau étant de "se reproduire lui-même", il est possible qu'une droite soit pour lui plus simple à reproduire qu'une courbe. Cela rendrait le réseau "rigide", réticent à changer certains de ses paramètres qui permettraient de gagner en précision sur l'énergie, et éventuellement en temps de calcul.

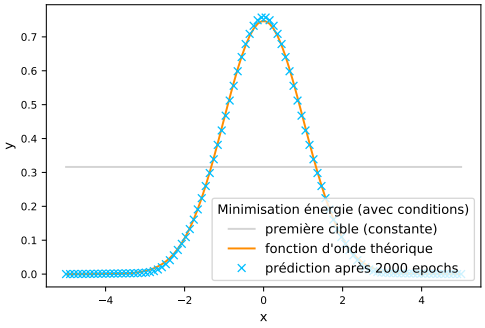

FIGURE 8: Version finale du programme. 2000 itérations, discrétisation sur 100 points.

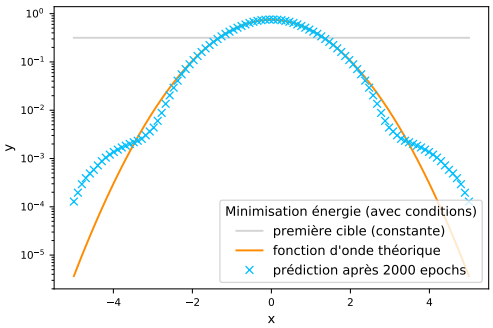

FIGURE 9: Version finale du programme. 2000 itérations, discrétisation sur 100 points. AU CHOIX AVEC LA FIGURE PRÉCÉDENTE

Les potentiels étudiés sur les figures 10 et 11 sont respectivement :

$$\begin{aligned} V(x) &= |x| \\ V(x) &= \frac{1}{2}x^2 + 3e^{-4x^2} \end{aligned} \tag{7}$$

J'ai fait le choix d'utiliser l'état fondamental de l'oscillateur harmonique comme première cible pour le réseau afin d'accélérer le processus de minimisation.

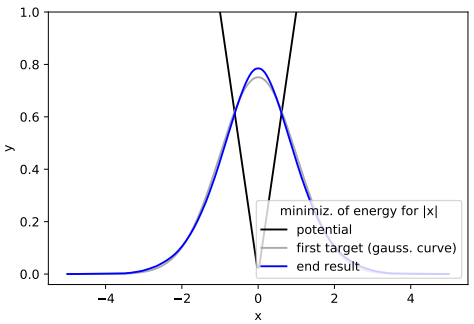

FIGURE 10: Discrétisation de l'espace sur 200 points et 7000 itérations.

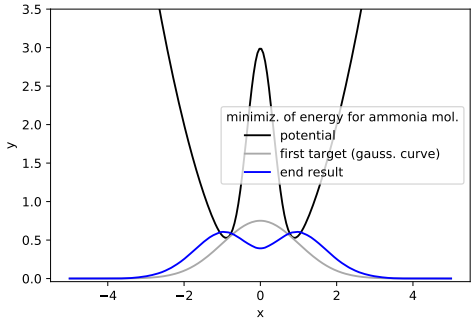

FIGURE 11: Discrétisation de l'espace sur 200 points et 7000 itérations.

Pour finir, la figure 12 représente le temps de calcul CPU en fonction de la géométrie utilisée. On y voit que les réseaux à 500 et 5000 paramètres ont des temps CPU similaires pour la plupart des géométries. 30600 est toujours un peu au dessus et même très au dessus pour le réseau à 1 couche.

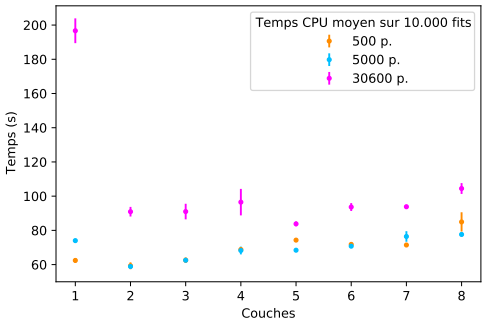

FIGURE 12: Temps CPU au bout de 10000 itérations, moyenné sur 30 runs.

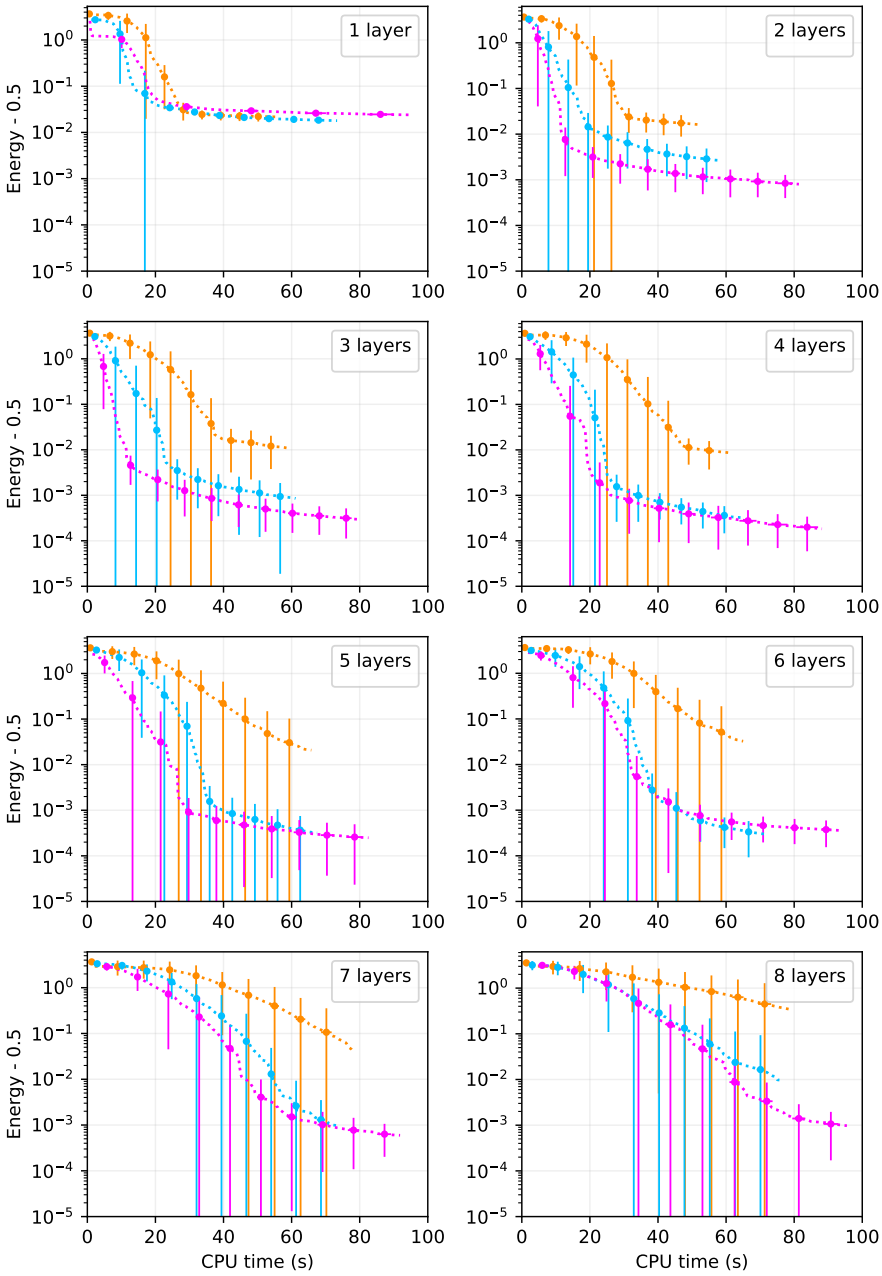

FIGURE 13: Orange : 500 paramètres ; Bleu : 5000 ; Rose : 30600. Les courbes s'arrêtent lorsque le programme a effectué 10000 itérations, à l'exception de "1 couche - rose" qui est coupée à 5000 itérations pour des raisons de lisibilité (200 secondes au total). La précision et le temps sont moyennés sur 30 runs.

4 Conclusion

Nous avons étudié deux méthodes numériques de résolution de l'équation de Schrödinger indépendante du temps permettant de trouver la fonction d'onde et l'énergie de l'état fondamental d'un oscillateur harmonique. Contrairement à un algorithme de diffusion tel que RK4 (page 1), le programme de minimisation stochastique ne permet de trouver que l'état fondamental pour le moment et il est encore tôt pour conclure sur les performances du programme du fait que les résultats de la figure 13 page 7 ne sont pas complètement convergés après 10000 itérations pour la plupart des géométries étudiées. On sait néanmoins que les réseaux de neurones peuvent être utilisés dans le cadre des méthodes variationnelles et de nouvelles questions se posent déjà : y a-t-il un nombre optimal de points pour la discrétisation de l'espace ? Quelles sont ses performances sur des temps plus longs ? Que peuvent apporter de nouvelles géométries ? Geler des couches dans la limite asymptotique permet-il d'accélérer la minimisation ? Comment se comporte le loss lorsqu'on change la cible en court d'apprentissage ? Quelles sont les performances du programme à deux et trois dimensions ?

Bibliographie

[1] Balázs Csanád Csáji. "Approximation with Artificial Neural Networks, MSc Thesis". In : *Eötvös Loránd University (ELTE), Budapest, Hungary* 24 :48 (2001). URL : https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.2647&rep=rep1&type=pdf.

[2] Claude Cohen-Tannoudji, Bernard Diu et Franck Laloë. *Mécanique quantique Tome 1*. EDP Sciences, 2018, p. 371-378. ISBN : 9782759822874.

[3] Colin Bernet. *Handwritten Digit Recognition with scikit-learn*. URL : https://thedatafrog.com/en/articles/handwritten-digit-recognition-scikit-learn/.

[4] Colin Bernet. *Le réseau à un neurone : régression logistique*. URL : https://thedatafrog.com/fr/articles/logistic-regression/.

[5] Colin Bernet. *Premier réseau de neurones avec keras*. URL : https://thedatafrog.com/fr/articles/first-neural-network-keras/.

[6] Google. *Colaboratory*. URL : https://colab.research.google.com/.

[7] Keras. *Model training*. URL : https://keras.io/api/models/model_training_apis/.

[8] Clément Lotteau. *Mon GitHub*. URL : https://github.com/quadrivecteur?tab=repositories.

[9] SciPy. *Integration avec SciPy*. URL : https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html.

[10] SciPy. *Interpolation avec SciPy*. URL : https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html.