Phase 2

clement lotteau

June 2020

Table des matières

1 Comparaison 'mse' de keras et custom loss avec tensor flow

```python
#CUSTOM LOSS
Loss = np.zeros((nb_epochs,runs))         #Array des pertes

def my_loss_fn(y_true, y_pred):
    squared_difference = tf.square(y_true - y_pred)
    return tf.reduce_mean(squared_difference, axis=-1)

for k in range(0,runs):
  #Approximation par machine learning
  model = models.Sequential([
      layers.Dense(200, input_shape=(1,), activation='relu'),
      layers.Dense(200, input_shape=(1,), activation='relu'),
      layers.Dense(1), # no activation -> linear function of the input
  ])
  model.summary()
  opt = optimizers.Adam(learning_rate=0.001)
  model.compile(loss=my_loss_fn,optimizer=opt)
  history = History()
  hist = model.fit(linx, norm,batch_size=50, epochs=nb_epochs,callbacks=[history])
  predictions = model.predict(linx)
  preds = predictions.reshape(-1)
  #Récupération du loss
  for i in range(0,nb_epochs):
    Loss[i,k] = hist.history['loss'][i]
```

FIGURE 1 – Code de la custom loss

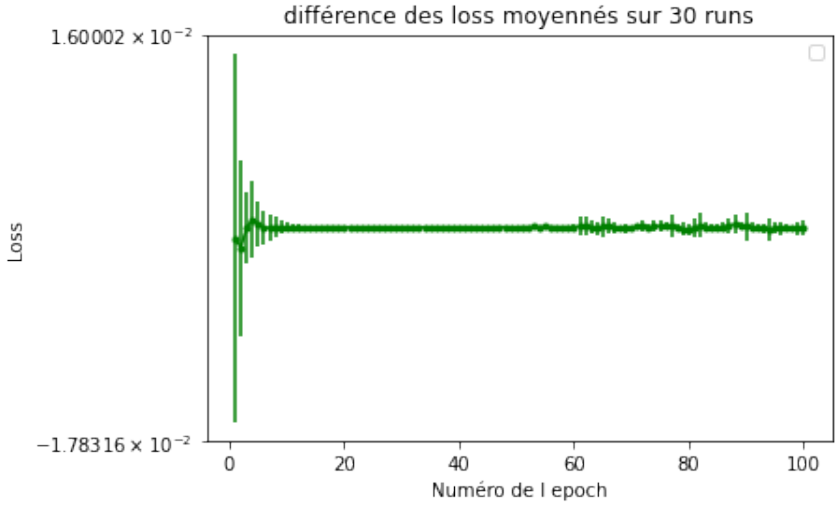
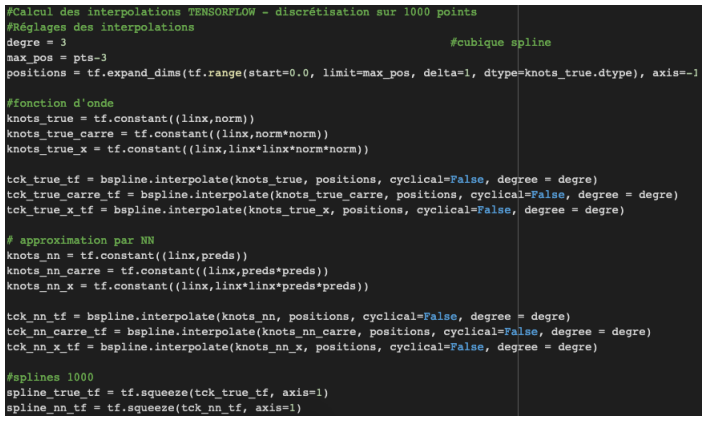


FIGURE 2 – Différence des loss moyennée sur 30 runs (mse - custom)

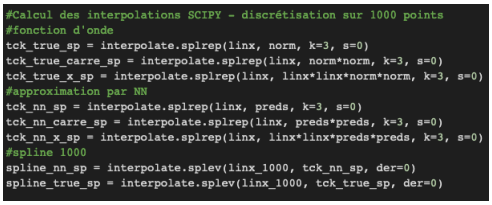
2 Comparaison des interpolations Scipy - Tensorflow

2.1 Code



```
#Calcul des interpolations TENSORFLOW - discrétisation sur 1000 points
#Réglages des interpolations
degre = 3                                                    #cubique spline
max_pos = pts-3
positions = tf.expand_dims(tf.range(start=0.0, limit=max_pos, delta=1, dtype=knots_true.dtype), axis=-1

#fonction d'onde
knots_true = tf.constant((linx,norm))
knots_true_carre = tf.constant((linx,norm*norm))
knots_true_x = tf.constant((linx,linx*linx*norm*norm))

tck_true_tf = bspline.interpolate(knots_true, positions, cyclical=False, degree = degre)
tck_true_carre_tf = bspline.interpolate(knots_true_carre, positions, cyclical=False, degree = degre)
tck_true_x_tf = bspline.interpolate(knots_true_x, positions, cyclical=False, degree = degre)

# approximation par NN
knots_nn = tf.constant((linx,preds))
knots_nn_carre = tf.constant((linx,preds*preds))
knots_nn_x = tf.constant((linx,linx*linx*preds*preds))

tck_nn_tf = bspline.interpolate(knots_nn, positions, cyclical=False, degree = degre)
tck_nn_carre_tf = bspline.interpolate(knots_nn_carre, positions, cyclical=False, degree = degre)
tck_nn_x_tf = bspline.interpolate(knots_nn_x, positions, cyclical=False, degree = degre)

#splines 1000
spline_true_tf = tf.squeeze(tck_true_tf, axis=1)
spline_nn_tf = tf.squeeze(tck_nn_tf, axis=1)
```

FIGURE 3 – code de l'interpolation de Tensorflow.



```
#Calcul des interpolations SCIPY - discrétisation sur 1000 points
#fonction d'onde
tck_true_sp = interpolate.splrep(linx, norm, k=3, s=0)
tck_true_carre_sp = interpolate.splrep(linx, norm*norm, k=3, s=0)
tck_true_x_sp = interpolate.splrep(linx, linx*linx*norm*norm, k=3, s=0)
#approximation par NN
tck_nn_sp = interpolate.splrep(linx, preds, k=3, s=0)
tck_nn_carre_sp = interpolate.splrep(linx, preds*preds, k=3, s=0)
tck_nn_x_sp = interpolate.splrep(linx, linx*linx*preds*preds, k=3, s=0)
#spline 1000
spline_nn_sp = interpolate.splev(linx_1000, tck_nn_sp, der=0)
spline_true_sp = interpolate.splev(linx_1000, tck_true_sp, der=0)
```

FIGURE 4 – code de l'interpolation de Scipy.

À noter : l'interpolation de Tensorflow enlève des points à la discrétisation selon x. Ce nombre de points enlevé correspond au degré de l'interpolation (cubique ici, donc 3 points). Il faut donc effectuer deux discrétisations et deux calculs de fonction d'onde pour comparer SciPy et Tensorflow.

2.2 Plot des interpolations SciPy et Tensorflow du N.N.)

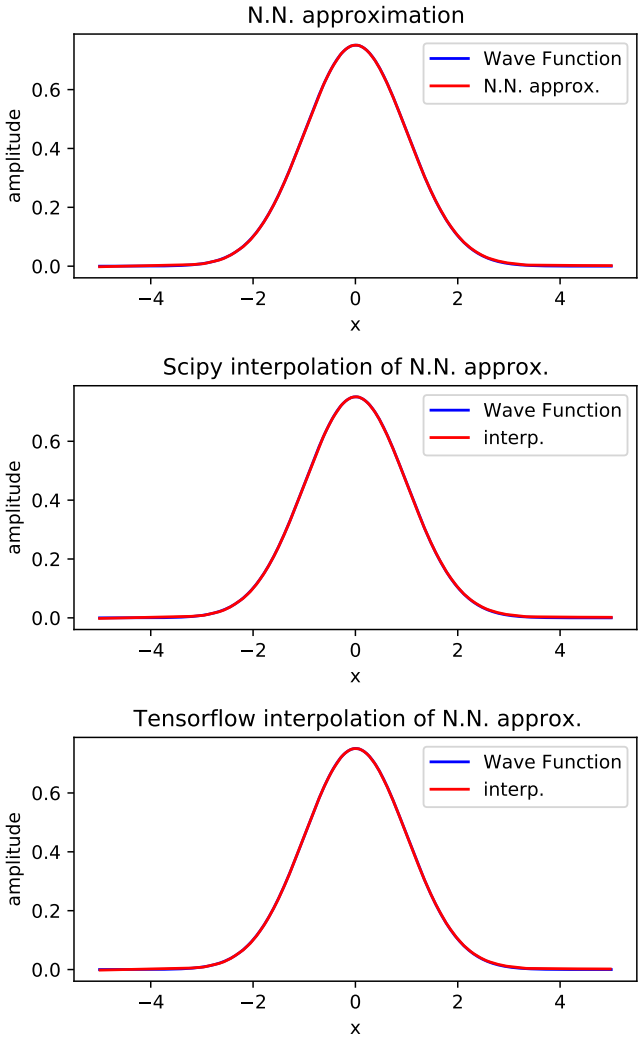


FIGURE 5 – Interpolation cubique sur 1000 points.

2.3 Difference des interpolations SciPy - Tensorflow (fonction d'onde)

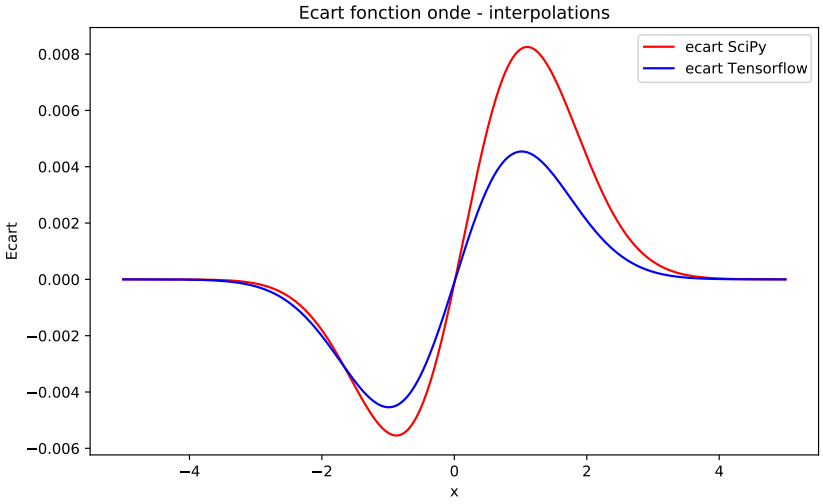


FIGURE 6 – Ecart entre la fonction d'onde et ses interpolations (pas de N.N.).

Comparaison en valeur absolue :
Ecart total/points (tf) : 0.0015
Ecart total/points (sp) : 0.0023

L'interpolation de Tensorflow donne un meilleur résultat que celui de Scipy pour la fonction d'onde.

Je vais maintenant calculer les écarts entre les interpolations du résultat par N.N. et la fonction d'onde.

2.4 Ecarts entre la fonction d'onde et le N.N., et ses interpolations.

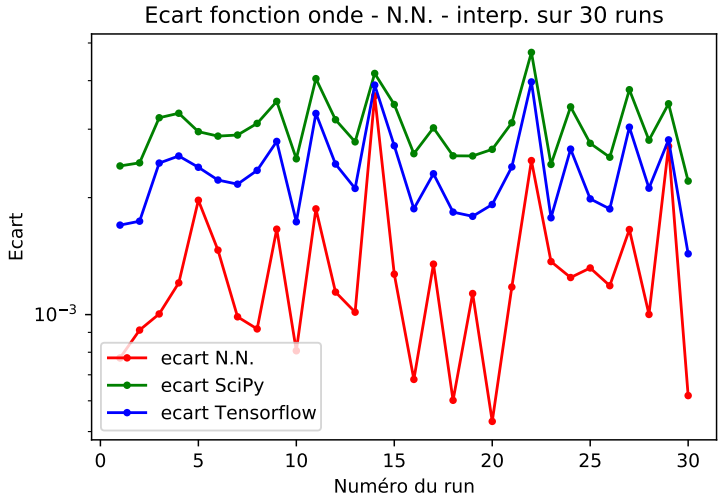


FIGURE 7 – Ecart total divisé par le nombre de points. 30 runs.

On confirme que les splines de Tensorflow sont meilleurs que ceux de SciPy.

3 Minimisation sans fonction de coût

Le but ici est de se servir des erreurs de fit pour sélectionner une prédiction comme nouvel objectif de fit si l'énergie de cette dernière est inférieure à la fonction à fitter. Le réseau "se fit lui-même". On note que le réseau est reconstruit à chaque fois qu'une nouvelle cible est choisie.

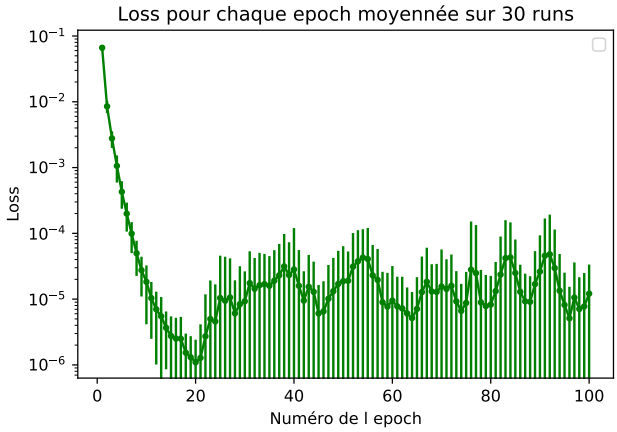


FIGURE 8 – Loss pour un réseau 200x200x200x200 (batch 50).

Je me sers des oscillations autour du minimum de la fonction de coût en 'croisant les doigts' pour que le réseau donne une prédiction ayant une énergie inférieur à celle de sa cible. Je remplace ensuite la cible par la prédiction. J'ai donc besoin d'un réseau qui minimise rapidement la fonction de coût. La précision de la minimisation de l'énergie dépend (à priori) de l'amplitude des oscillations autour du minimum. De grandes oscillations sont (à priori) bonnes pour une première approximation, et de petites oscillations permetteraient peut-être d'affiner la minimisation.

```
#INITIALISATION DU MODEL
model = models.Sequential([
  layers.Dense(200, input_shape=(1,), activation='relu'),
  layers.Dense(200, input_shape=(1,), activation='relu'),
  layers.Dense(1), # no activation -> linear function of the input
])
model.summary()
opt = optimizers.Adam(learning_rate=0.001)
model.compile(loss=min_loss,optimizer=opt)


for i in range(0,1001):

  #fit de l'onde
  model.fit(linx,onde,epochs=1,batch_size=50)
  predictions = model.predict(linx)
  preds = predictions.reshape(-1)


  #sélection de l'onde avec l'énergie la plus faible
  if (calcul_energie(linx,preds) < calcul_energie(linx,onde)):
    print('Energie onde = ',calcul_energie(linx,onde))
    print('Energie preds = ',calcul_energie(linx,preds))
    onde = preds

    #clear et recréation du modèle
    keras.backend.clear_session()
    model = models.Sequential([
      layers.Dense(200, input_shape=(1,), activation='relu'),
      layers.Dense(200, input_shape=(1,), activation='relu'),
      layers.Dense(1), # no activation -> linear function of the input
    ])
    model.summary()
    opt = optimizers.Adam(learning_rate=0.001)
    model.compile(loss=min_loss,optimizer=opt)
```

FIGURE 9 – premier code de la minimisation sans passer par la fonction de coût.

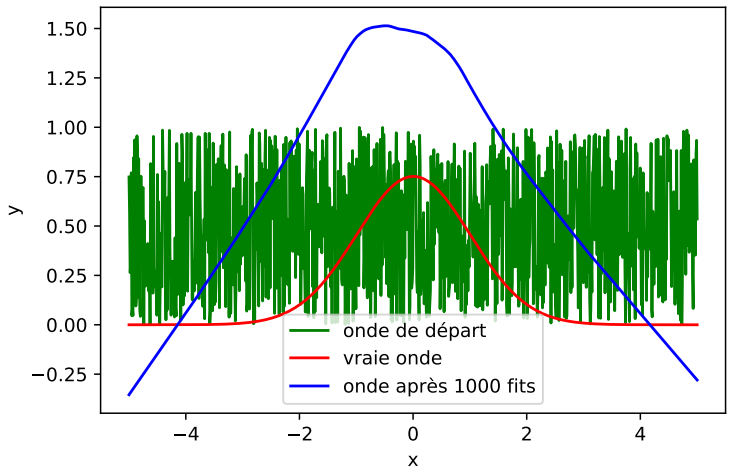


FIGURE 10 – Réseau 200x200.

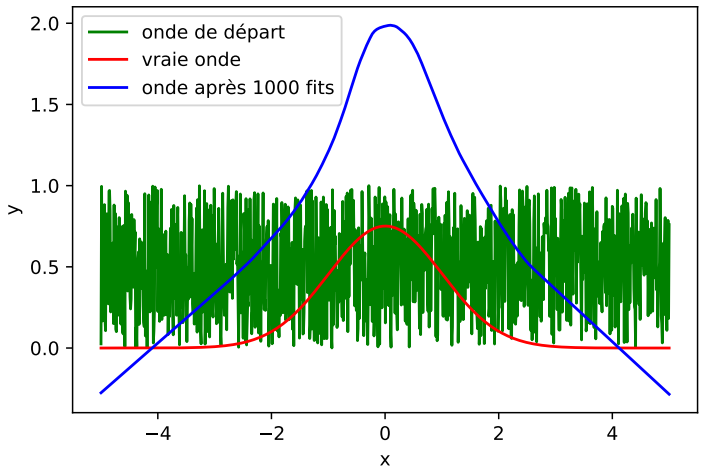


FIGURE 11 – Réseau 200x200x200.

Je dois imposer à la fonction d'onde d'être symétrique, toujours positive et normée à 1.

3.1 symétrique, positive et normée à 1

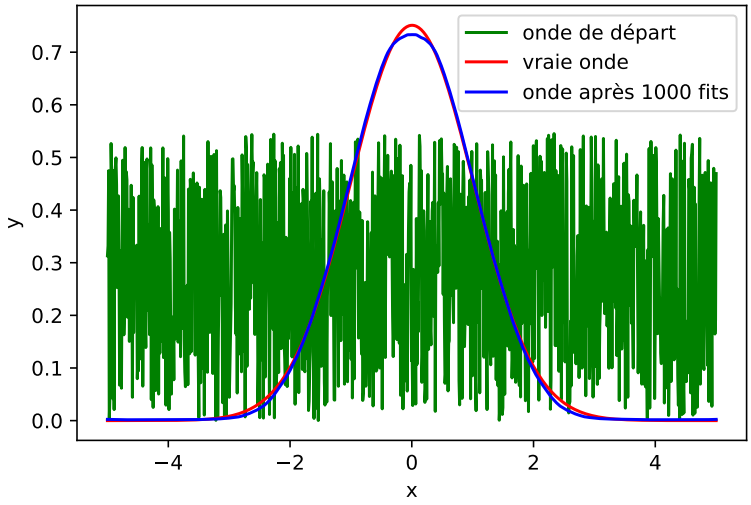


FIGURE 12 – 7 couches, 30671 paramètres. 71x71x71x71x71x70x70. Energie = 0.50258 (0.50001 à trouver).

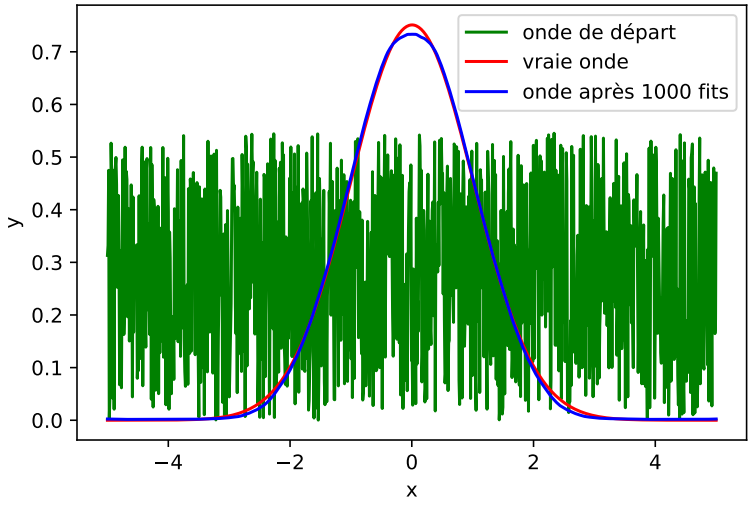


FIGURE 13 – 6 couches, 30574 paramètres. 78x78x78x77x77x76. Energie = 0.50200 (0.50001 à trouver)

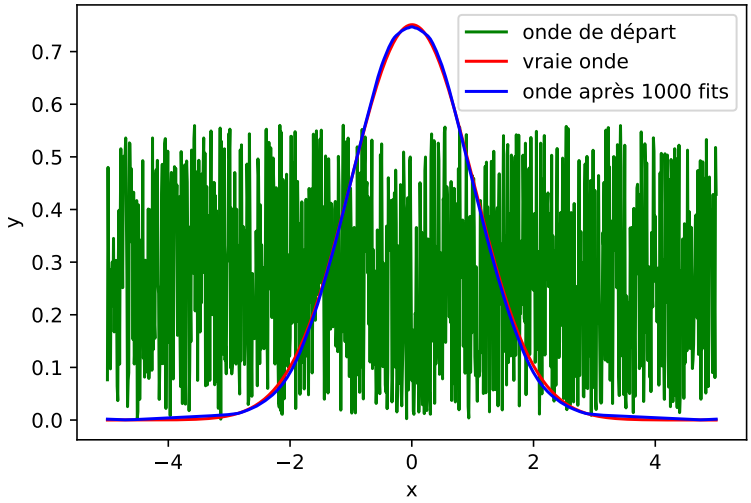


FIGURE 14 – 5 couches, 30623 paramètres. 87x87x87x86x86. Energie = 0.50215 (0.50001 à trouver)

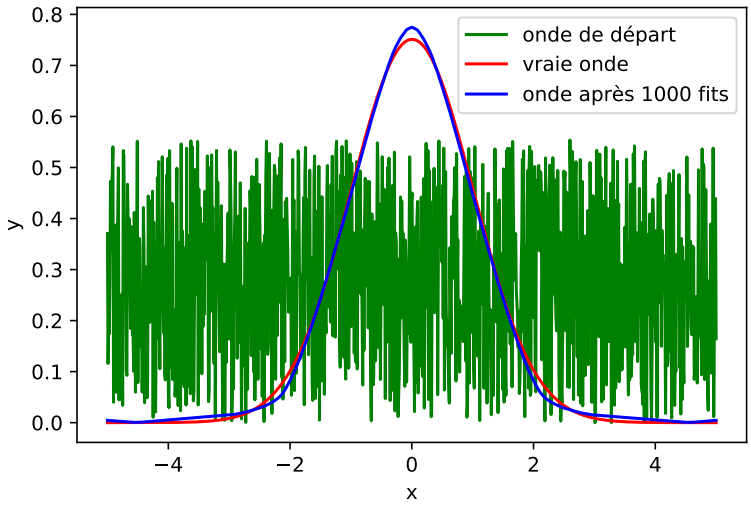


FIGURE 15 – 4 couches, 30601 paramètres. 100x100x100x100. Energie = 0.50572 (0.50001 à trouver)

3.2 Sans reconstruction au delà d'un seuil de l'énergie

La reconstruction du réseau sert à oublier les apprentissages précédents et à accélérer le passage d'une onde "plate" à une gaussienne. Beaucoup de temps est perdu à cause de cette reconstruction lorsque la fonction a déjà la forme d'une gaussienne. J'ai donc placé un seuil à une énergie de 0.6 en dessous de laquelle le réseau cesse de se reconstruire et se contente de fitter en boucle 1000 fois. Un seuil de 1 donnait des résultats moints concluants (la gaussienne était aplatie en haut).

J'ai aussi symétrisé et normalisé la fonction d'onde cible, ainsi que recalculé son énergie. Pareil pour la fonction d'onde aléatoire.

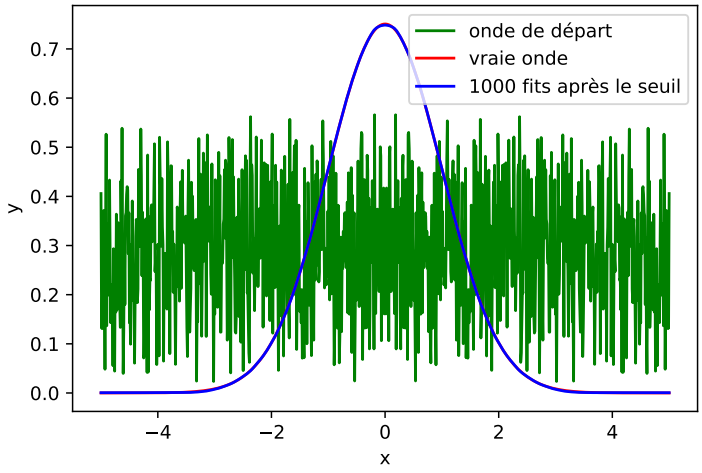


FIGURE 16 – 4 couches, 30601 paramètres. 100x100x100x100 (avant et après le seuil). Seuil à 0.6

Energie = 0.50033 (0.500001 à trouver). Je n'arrive pas à descendre vraiment plus en dessous de cette énergie.

3.3 100 points, sans reconstruction, sans seuil

J'ai abandonné la fonction d'onde aléatoire au profit d'une fonction d'onde constante et normalisée.

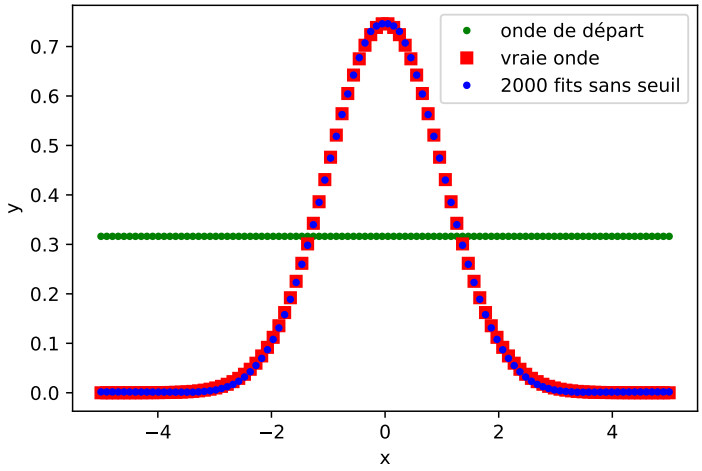


FIGURE 17 – 4 couches, 30601 paramètres. 100x100x100x100. 2000 fits.

Energie = 0.5005 (0.5001 à trouver). Temps de calcul : 18.3 secondes

Il serait intéressant de regarder l'écart en fonction de x. Il serait peut-être possible de minimiser cet écart avec des biais dans le réseau aux endroits où le fit est le moins bon (à vue d'oeil : souvent où la courbure est forte).