TP2

Le TP est dédié à l'initialisation de CNNs à p cachées avec ReLU, pour retrouver des fonctions dans la classe de Takagi telle que $x \mapsto x^2$. Plus généralement on considérera des fonctions du type

$$f(x) = \sum_{n=1}^{p} \frac{1}{a^n} g_n(x) \tag{1}$$

La structure d'un script adapté est détaillée ci-dessous:

```
from keras import backend as K

#declaration des parametres des couches en fonction fac=a et depth=p
fac=4; depth=2
def init_W0(shape, dtype=None): return K.constant(np.array([[1,1,0]]))
def init_b0(shape, dtype=None): return K.constant(np.array([0,-0.5,0]))
def init_W1(shape, dtype=None):
W= np.array([[2,2,2/fac],[-4,-4,-4/fac],[0,0,1]]);return K.constant(W)
def init_W2(shape, dtype=None):
W= np.array([[2,2,2/fac**2],[-4,-4,-4/fac**2],[0,0,1]]); return K.constant(W)
def init_W3(shape, dtype=None):
W=np.array([[2/fac**depth],[-4/fac**depth],[1]]); return K.constant(W)

#declaration et initialisation du CNN
model = Sequential()
model.add(Dense(3, input_dim=1,name="lay0",kernel_initializer=init_W0,
use_bias=True,bias_initializer=init_b0,activation='relu'))
model.add(Dense(3, input_dim=1,name="lay1",kernel_initializer=init_W1,
use_bias=True,bias_initializer=init_b1,activation='relu'))
model.add(Dense(3, input_dim=1,name="lay2",kernel_initializer=init_W2,
use_bias=True,bias_initializer=init_b2,activation='relu'))
model.add(Dense(3, input_dim=1,name="lay3",kernel_initializer=init_W3,
use_bias=False,bias_initializer=init_b3,activation='relu'))

# affichage fonction resultante
x_p=np.linspace(0,1,100); y_p=model.predict(x_p);
plt.plot(x_p,y_predict_ini)
```

a) Analyser l'initialisation des (W_i, b_i) et vérifier que cela correspond à fonction (**??**).

b) Charger `https://github.com/despresbr/NNNA/blob/main/TP3.py` et exécuter pour afficher la fonction f.

c) Désactiver l'instruction `sys.exit(0)` en ligne 102.
Vérifier que la fonction est construite par le NN est proche de $f(x) = x - x^2$.
Mesurer l'erreur en norme L^2 discrète entre $f(x) = x - x^2$ et sa reconstruction NN avec ReLU et p couches cachées.

d) Evaluer l'ordre de grandeur de l'erreur en fonction du nombre de neurones. Cela est-il en accord avec la théorie?

e) A présent nous allons effectuer une phase de training. Charger `https://github.com/despresbr/NNNA/blob/main/TP3_data.py` et exécuter pour générer les données.

f) Désactiver l'instruction `sys.exit(0)` en ligne 250.
Effectuer un training pàrtir d'un NN à 3 couches.
Vérifier que la précision initiale est préservée: cela indique la **stabilité** de ce NN.

g) Enfin remplacer tous les initialiseurs par `'random_uniform'`. Refaire un training et vérifier que la précision est perdue.
La conclusion générale les coefficients des fonctions de la classe de Takagi ne sont pas calculées naturellement par les NN+training, ou en tout cas qu'elles sont **non attractives** pour la séance de training.