TP5/TP6

On considère le problème de calcul scientifique de la reconstruction de paramètres d'une interface droite

$$D = \{(x, y) \mid \cos\theta x + \sin\theta y = d\}.$$

Ici on a un patch de $(2N+1) \times (2N+1)$ cellules carrées unitiés d'aire chacune égale à 1, avec $N = 1$.

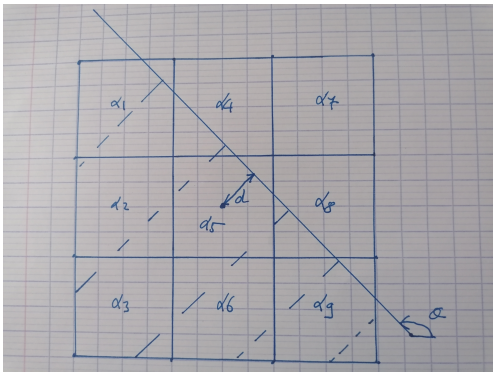

Figure 1: $\alpha_2 = \alpha_3 = \alpha_6 = 1$ et $\alpha_7 = 0$.

Les fractions de volume son

$$0 \leqslant \alpha_i = \int_{C_i \cap = \{a+bx+cy<0\}} dxdy \leqslant 1.$$

Le problème est de reconstruire θ à partir de la connaissance de α_i $1 \leqslant i \leqslant (2N+1)^2$. Plus précisément, on cherchera à reconstruire un réseau de neurones qui réalise

$$\begin{array}{rccc} f: & \mathbb{R}^9 & \to & \mathbb{R}^2 \\ & (\alpha_i)_{i=1}^0 & \mapsto & (\cos\theta, \sin\theta). \end{array}$$

. Nous allons voir comment créer cette fonction en Python/Keras/Tensorflow, puis comment exporter cette fonction dans un code C++.

- Charger `~despres/interface_create.py`, et analyser comment le soft construit par intégration numérique un dataset de type

$$\mathcal{D} = \{(\cos\theta, \sin\theta, \alpha_1, \ldots, \alpha_9) \text{ pour une variété de } (\theta, d)\}.$$

Bien vérifier que l'échatillonage en angle est tous les demi-degrés.

- Charger `~despres/interface_learn.py`, et analyser comment le soft "apprend"⇔faire du fit non-linéaire⇔fait de l'optimisation numérique pour construire f. Essayer d'obtenir une précision "moyenne" de l'odre du 1%.

  Activer les lignes

  ```
  data = np.array([[1,1,1,0.5,0.5,0.5,0,0,0]])
  prediction = model.predict(data)
  print("prediction=",prediction)
  ```

  et interpréter le résulat.

- Pour exporter le modèle en C++, on propose d'utiliser le soft *keras2cpp* `https://github.com/gosha20777/keras2cpp` du à Georgy Perevozchikov.

  Installer keras2cpp sur votre ordinateur et faire l'example (il faut copier `example.model` dans *build*).

  ```
  $ git clone https://github.com/gosha20777/keras2cpp.git
  $ cd keras2cpp
  $ mkdir build && cd build
  $ python3 ../python_model.py
  [[-1.85735667]]

  $ cmake ..
  $ cmake --build .
  $ cp ../example.model .
  $ ./keras2cpp
  [ -1.857357 ]
  ```

  Charger `~despres/cpp_interface.cc`, modifier la dernière ligne du fichier CMakeLists.txt, relancer la compilation, copier `interface.model` dans *build*, et lancer l'exécutable. Comparer le résultat avec la prédiction du point précédent.

- Reprendre la chaine de calcul du début et travailler avec des patchs 5×5. Comparer les résultats (si possible).

- A présent on s'intéresse à l'implémentation d'un réseau de neurones qui implémente $x \mapsto x^3$.

  Détailler la mise à plat (instruction *Flatten*) et comprendre en quoi cela permet une implémentation récursive.

  Pour les courageux et les imaginatifs, implémenter avec un niveau quelconque de récursivité.

Algorithm 1 Implementation of $x \mapsto x^3$

```
Python-Keras-Tensorflow Initialization
def init_W0_T(shape,dtype=None):    return K.constant(np.array([[1,2,-2,2 ]]))
def init_b0_T(shape,dtype=None):    return K.constant(np.array([0,0,2, -1 ]))
def init_T_W_e0(shape,dtype=None):  return K.constant(np.array([[3./2.]]))
def init_T_b_e0(shape,dtype=None):  return K.constant(np.array([-3./4. ]))
def init_T_W_e00(shape,dtype=None): return K.constant(np.array([[1]]))
def init_T_b_e00(shape,dtype=None): return K.constant(np.array([-1./8.]))
def init_W_out(shape, dtype=None):
 W=np.array([[1],[1./8.],[1./8.],[1./4.],
             [1*0],[1./8.*1./8.],[1./8.*1./8.],[1./4.*1./8.],
             [1*0.],[1./8.*1./8.],[1./8.*1./8.],[1./4.*1./8.],
             [1*0.],[1./8.*1./4.],[1./8.*1./4.],[1./4.*1./4.],  ])
return K.constant(W)
model = Sequential()
model.add(Dense(4, input_dim=1, kernel_initializer=init_W0_T,
                use_bias=True, bias_initializer=init_b0_T, activation=T_relu))
model.add(Reshape((4,1)))
model.add(Dense(4, input_dim=1, kernel_initializer=init_W0_T,
                use_bias=True, bias_initializer=init_b0_T, activation=T_relu))
model.add(Flatten()); model.add(Reshape((16,1)))
model.add(Dense(1, input_dim=1, kernel_initializer=init_T_W_e0,
          use_bias=True, bias_initializer=init_T_b_e0,  activation=T_relu))
model.add(Dense(1, input_dim=1,              kernel_initializer=init_T_W_e00,
          use_bias=True, bias_initializer=init_T_b_e00,  activation='linear'))
model.add(Flatten())
model.add(Dense(1,kernel_initializer=init_W_out,
          use_bias=False, activation='linear'))
n=100; x_p=np.linspace(0,1,n); y_i=model.predict(x_p); plt.plot(x_p,y_p)
```