TP n°2: Descente de gradient

♦ Exercice 1

Partie 1 Pour effectuer une descente de gradient, il est nécessaire de calculer la valeur du gradient à chaque étape. Pour ce faire, le cadriciel TensorFlow propose l'utilisation d'un ruban de gradients. L'utilisation est simple, lorsque les opérations sont effectuées à l'aide de tenseurs, `TensorFlow` calcule automatiquement la valeur du gradient au point considéré.

1. Écrire une fonction qui détermine le minimum de la fonction $(x;y) \longmapsto 4(x-1)^2 + \frac{1}{2}(e^{y-3}-1)^2$ par descente de gradient dont le prototype est le suivant :

   Fonction descenteGradient($\mathbf{x}_0$:*couple de flottants*, maxIter:*entier*, pas:*flottant*, ε:*flottant*) Ⓡ $\mathbf{x}$:*ndarray (2,1)*,*n*:*entier*

   Début

   Renvoie $\underset{\mathbb{R}^2}{\operatorname{argmin}}\, 4(x-1)^2 + \frac{1}{2}(e^{y-3}-1)^2$

   et $\underset{[\![1;maxIter]\!]}{\min} \|\overrightarrow{\nabla}(\mathbf{x}_n)\| < \varepsilon$ s'il existe n tel que $\|\overrightarrow{\nabla}(\mathbf{x}_n)\| < \varepsilon$ et maxIter sinon

   Fin

2. Déterminer le pas constant optimal
3. Vérifier numériquement

Memo TensorFlow

- Charger la classe TensorFlow

```
import tensorflow as tf
```

- Créer un tenseur variable de dimension 1, initialisé avec la valeur 2

```
x0=tf.Variable(2.0,name='x')
```

- Calculer le nombre dérivé en 2 de la fonction sigmoïde.

```
with tf.GradientTape() as tape:
  y=tf.math.sigmoid(x0)
tape.gradient(y,[x0])
```

- La fonction exponentielle

```
tf.math.exp()
```

- Calculer la norme ℓ_2 de $\mathbf{x}$

```
from numpy.linalg import norm
norm(x,2)
```

Partie 2 Méthode du moment

4. Proposer une seconde version de la fonction précédente où la formule de récurrence est :

$$\mathbf{x}_{n+1} = \mathbf{x}_n + m(\mathbf{x}_n - \mathbf{x}_{n-1}) - p\overrightarrow{\nabla} f(\mathbf{x}_n)$$

   Le vecteur $\mathbf{x}_n - \mathbf{x}_{n-1}$ s'appelle le moment et donne de l'inertie à la descente de gradient; la rendant moins sensible aux changements de variation.

5. Trouver une valeur de l'hyperparamètre m qui améliore la méthode de descente de gradient classique.
6. Yurii Nesterov a proposé d'optimiser la méthode du moment en évaluant le gradient non pas en $\mathbf{x}_n$ mais en $\mathbf{x}_n + m(\mathbf{x}_n - \mathbf{x}_{n-1})$. En remarquant qu'*a priori* $\mathbf{x}_n + m(\mathbf{x}_n - \mathbf{x}_{n-1})$ est plus proche de la solution que $\mathbf{x}_n$

   Écrire une troisième version de la fonction, comparer et commenter.

TP n°2: Descente de gradient

◆ **Exercice 2** Le fichier `isolation.csv` est une extraction des données présentes sur le site de ADEME à l'adresse suivante : https://data.ademe.fr/datasets/isolation.

1. Charger les données dans un dataframe `pandas`, proposer des représentations adaptées des données. Deux colonnes sont quasiment redondantes, lesquelles ? Justifier.
2. On souhaite inférer la variable cout_total_ht à partir des variables resistance et surface à l'aide d'une régression linéaire.
   (a) Écrire un programme en python pour déterminer les coefficients de la régression linéaire par la méthode de descente de gradient. On utilisera la différentiation automatique. On pourra également tester les variantes vues à l'exercice précédent.
   (b) Étudier la validité du modèle.

Memo TensorFlow/scikit-learn

- Initialiser aléatoirement un tenseur de taille (3; 1)

```
a=tf.Variable(tf.random.normal((3,1)),name='a')
```

- Centrer et réduire les données

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit_transform(Data)
```

- Matrice de corrélation d'un dataframe `pandas`

```
matrice.corr()
```

- Opérateur du produit matriciel

```
x@a
```

- Calculer la moyenne d'un tenseur

```
tf.reduce_mean()
```