

## § Python et Machine Learning §

### Problem 1: Exercice 1.103 - Jeu des 3 boîtes

L'animateur d'un jeu presente trois boîtes fermées a un candidat. Une de ces boîtes contient un chèque permettant au candidat de remporter un prix et les autres sont vides, la boîte victorieuse étant désignée par un tirage au sort équiprobable et seul l'animateur ayant connaissance du résultat de ce tirage. Le jeu se déroule de la manière suivante :

1. l'animateur demande au candidat de choisir une de ces boîtes, sans l'ouvrir;
2. l'animateur ouvre alors une des deux boîtes restantes, mais jamais celle contenant le chèque (si aucune des deux ne contient le chèque, l'animateur ouvre au hasard une des deux boîtes);
3. l'animateur propose alors au candidat d'éventuellement changer son choix de la boîte à ouvrir, parmi les deux restantes;
4. l'animateur ouvre la boîte désignée par le candidat, qui remporte ou non le prix selon que la boîte ouverte contient ou non le chèque.

Le problème consiste à déterminer la meilleure stratégie du candidat parmi les deux possibles :

S1 : rester sur son choix initial, la probabilité que le chèque se trouve dans l'une des deux boîtes restantes étant alors considérée par le candidat comme être de  $1/2$  pour chaque boîte;

S2 : modifier son choix initial, la probabilité que le chèque se trouve dans la boîte initialement choisie étant alors considérée par le candidat comme être plus faible que celle que le chèque se trouve dans l'autre boîte.

**(1) Ecrivez une fonction qui simule toutes les étapes de ce jeu avec en paramètre la stratégie choisie par le candidat et qui renvoie 1 si le joueur gagne, 0 sinon. Utilisez pour les tirages aléatoires la bibliothèque random et implémentez les différents états du jeu sous la forme de listes de numéros de boîtes (par exemple la liste des deux boîtes restantes une fois le premier choix du candidat effectuée). Pour contrôler le déroulement du jeu, affichez les résultats des différentes étapes lorsqu'une variable booléenne globale DEBUG est vraie. Voici le code de notre fonction play :**

Code Python question 1

```
1 def play(strategy, DEBUG):
2     if DEBUG:
3         print("#####")
4         if strategy=="s1":
5             print("Strategie 1 : Le joueur reste sur son premier choix")
6         else:
7             print("Strategie 2 : Le joueur choisi une des deux îbotes restante")
8     boxes = [1, 0, 0]
9     boxes = random.sample(boxes, len(boxes))
10    first_move = boxes[random.randint(0, len(boxes)-1)]
11    if DEBUG:
12        print(boxes)
13        if first_move == 1:
14            print("Le joueur a choisi la boîte contenant le chèque : first_move =",
15                  ↪ first_move)
16        else:
17            print("Le joueur a choisi une boîte vide : first_move =", first_move)
18    boxes.remove(first_move)
19    if DEBUG:
20        print("Les boîtes restantes sont :", boxes)
21        print("#####\n")
```

```
21 if strategy == "s2":
22     if first_move == 1:
23         second_move = boxes[random.randint(0, 1)]
24     else:
25         second_move = 1
26     if DEBUG:
27         if second_move == 1:
28             print("Le joueur a choisi une des deux boites contenant le cheque :
29                 ↪ second_move =", second_move)
30         else:
31             print("Le joueur a choisi une boite vide : second_move =",
32                 ↪ second_move)
33             print("#####\n")
34 if strategy=="s1" and first_move == 1:
35     return 1
36 elif strategy=="s2" and second_move == 1:
37     return 1
38 else:
39     return 0
play("s2", True)
```

La sortie est la suivante :

```
In [5]: play("s2", True)
#####
Strategie 2 : Le joueur choisi une des deux boites restante
[1, 0, 0]
Le joueur à choisi la boîte contenant le chèque : first_move = 1
Les boîtes restantes sont : [0, 0]
#####

Le joueur à choisi une boîte vide : second_move = 0
#####

Out [5]: 0
```

Figure 1: Sortie de la def play avec la strategie 2 et toutes les etapes du jeu

La fonction play qu'on l'on a cree ici, reproduit le jeu des trois boîtes. Le choix de la boîte par le candidat ainsi que la boîte gagnante est choisi aleatoirement grâce a la bibliothèque random. La fonction possède deux paramètres, tout d'abord strategy qui devra être « s1 » ou « s2 » et qui correspond respectivement a « Le joueur garde sa boîte » et « Le joueur change de boîte ». Et ensuite le paramètre DEBUG qui s'il est vrai affichera toutes les etapes du jeu, et s'il est faux affichera seulement le resultat. En plus de retourner toutes les etapes du jeu si DEBUG est vrai, la fonction retourne 1 si le joueur gagne et 0 s'il perd.

(2) Ecrivez une fonction qui simule N fois le jeu, avec comme second paramètre la stratégie choisie, et qui affiche à la fin et avec 3 décimales la moyenne (fréquence) des coups gagnants pour cette stratégie-la. Voici le code de notre fonction playNtime :

Code Python question 2

```

1 def play_N_times(n, strategy, DEBUG):
2     results = []
3     for i in range(n):
4         results.append(play(strategy, DEBUG))
5     print("Le nombre de victoire moyen en utilisant la strategie", strategy, "est : "
6           ↪ ,np.round(statistics.mean(results), 3), "\n")
7
8 play_N_times(3, "s1", True)

```

La sortie est la suivante :

```

In [8]: play_N_times(3, "s1", True)
#####
Strategie 1 : Le joueur reste sur son premier choix
[1, 0, 0]
Le joueur à choisi la boîte contenant le chèque : first_move = 1
Les boîtes restantes sont : [0, 0]
#####

#####
Strategie 1 : Le joueur reste sur son premier choix
[0, 1, 0]
Le joueur à choisi la boîte contenant le chèque : first_move = 1
Les boîtes restantes sont : [0, 0]
#####

#####
Strategie 1 : Le joueur reste sur son premier choix
[1, 0, 0]
Le joueur à choisi une boîte vide : first_move = 0
Les boîtes restantes sont : [1, 0]
#####

Le nombre de victoire moyen en utilisant la stratégie s1 est : 0.667

```

Figure 2: Sortie de la def playNtime avec 3 simulation de la stratégie 1 et toutes les étapes du jeu

On a créé une fonction playNtimes qui permet de simuler ce jeu N fois. Cette fonction comporte trois paramètres, tout d'abord n qui correspond au nombre de fois que l'on souhaite simuler le jeu, et ensuite strategy et DEBUG les deux mêmes paramètres que pour la fonction play. La fonction retourne tout simplement le nombre de victoire moyen sur les N parties simulées.

(3) Ecrivez un programme qui simule N fois ce jeu pour chacune des deux stratégies. Faites-le tourner en mode DEBUG pour N = 5 puis sans le mode DEBUG pour N = 100000. Quels résultats obtenez-vous dans ce dernier cas ? Voici le code de notre fonction play each strats :

Code Python question 3

```

1 def play_each_strats(n, DEBUG):
2     play_N_times(n, "s1", DEBUG)
3     play_N_times(n, "s2", DEBUG)
4
5 play_each_strats(5, True)
6 play_each_strats(100000, False)

```

La sortie est la suivante :

```
In [10]: play_each_strats(100000, False)
Le nombre de victoire moyen en utilisant la stratégie s1 est : 0.334
Le nombre de victoire moyen en utilisant la stratégie s2 est : 0.666
```

Figure 3: Sortie de la def play each strats avec 100 000 simulation et sans les etapes du jeu

La dernière fonction que l'on a créée est `playeachstrats`, celle-ci permet de simuler  $N$  fois le jeu pour chacune des deux stratégies. La fonction possède donc deux paramètres  $n$  et `DEBUG` comme pour `playNtime`, sans le paramètre `strategy` évidemment car on simule ici les deux stratégies. Cette fonction retourne comme pour `playNtime` le nombre de victoire moyen sur les  $N$  parties simulées, mais pour les deux stratégies.

En lançant plusieurs fois la fois pour  $n=100000$ , on s'est rendu compte que pour la stratégie 1 le nombre de victoire moyen converge vers 0,333. Et pour la stratégie 2 le nombre de victoire moyen converge lui vers 0,666.

(4) On veut maintenant calculer les probabilités de gagner au jeu pour chacune des deux stratégies et les comparer aux fréquences obtenues par simulation. On définit comme ensemble des résultats possibles de l'expérience aléatoire l'univers  $\Omega = (b_0, b_1, b_2)$ , l'évènement élémentaire  $b_i$  signifiant que le candidat choisit initialement la boîte numéro  $i$  ( $i = 0, 1, 2$ ). Par convention, on suppose que la boîte qui contient le chèque porte le numéro 0. On définit en outre les évènements  $V$  : le candidat choisit initialement une des deux boîtes vides;  $Gs_1$  : le candidat gagne en suivant la stratégie  $S_1$ ;  $Gs_2$  : le candidat gagne en suivant la stratégie  $S_2$ . (i) Déterminez  $P(b_i)$   $\forall i = 0, 1, 2$  ainsi que  $P(V)$ . (ii) Déterminez  $P(Gs_1|V)$  et  $P(Gs_1|b_0)$ . (iii) Déduisez-en  $P(Gs_1)$ . (iv) Déterminez  $P(Gs_2|V)$  et  $P(Gs_2|b_0)$ . (v) Déduisez-en  $P(Gs_2)$ . (vi) Comparez et analysez les résultats de la simulation avec ceux théoriques. On a calculé les probabilités :

(i)  $P(b_i)$  pour tout  $i = 0, 1, 2 = 1/3$

$P(V) = 2/3$

(ii)  $P(Gs_1|V) = 0$

$P(Gs_1|b_0) = 1$

(iii)  $P(Gs_1) = 0 \cdot 2/3 + 1 \cdot 1/3 = 1/3$

(iv)  $P(Gs_2|V) = 1$

$P(Gs_2|b_0) = 0$

(v)  $P(Gs_2) = 1 \cdot 2/3 + 0 \cdot 1/3 = 2/3$

(vi) En comparant les résultats théoriques et les résultats de la simulation on se rend compte qu'ils sont similaires, et qu'il étaient donc prévisibles.

## Problem 2: Exercice 4.102 - Classification de chiffres manuscrits MNIST

Cet exercice propose l'étude d'un CNN pour classifier des images de chiffres manuscrits, en utilisant la base de donnée MNIST composee de 70000 imageries a niveaux de gris.

### (1) Quelle est la structure des tenseurs DA, Da, DT, Dt et que representent-ils ?

Les tenseurs DA, Da, DT et Dt representent l'ensemble des images d'entraînement et de test ainsi que les labels, les dimensions sont les suivantes : DA : (60000, 28, 28, 1), 60000 images de 28\*28 pixels et 1 pour le canal de couleurs (1 seul car niveau de gris). Da : (60000,) sont les labels des images d'entraînement. DT : (10000, 28, 28, 1), 10000 images de 28\*28 pixels en niveau de gris pour tester le modèle. Dt : (10000,) sont les labels des images de test.

(2) Rajoutez après le commentaire Cf. question 2) un code visualisant, pour un indice a saisir par l'operateur, l'image dans la base d'apprentissage ainsi que la valeur de la sortie associee a cette image. En entrant le code ci dessous, nous pouvons observer un element de la base de test.

Code Python question 2

```

1 # Cf. question 2)
2 fig = plt.figure(figsize=(10, 5))
3 question = "Entrez un indice entre 0 et", len(DA)
4 indice = input(question)
5
6 # Plot de l'image
7 plt.title('Le chiffre est {label}'.format(label=Da[int(indice)]))
8 plt.imshow(DA[int(indice)], cmap=plt.get_cmap('gray'))
9 plt.show()

```

La sortie est la suivante :

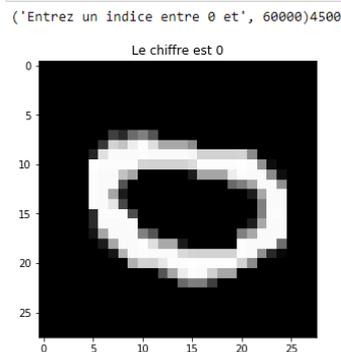


Figure 4: Affichage de la 4500eme image de la base d'entraînement et de son label

(3) Que realisent les lignes sous le commentaire cf. question 3) (i) Le code en dessous du commentaire permet de formater les images d'entrée pour placer les canaux de couleurs en premier (channels first) ou en dernier (channels last) en fonction de comment le backend va les lire. Ici, sachant que DA est de la forme (60000, 28, 28, 1), nous serons dans le cas "channel last" et de ce fait on reshape les dimensions des images pour y correspondre.

Même question pour les lignes sous le commentaire cf. question 3) (ii) On divise les images par 255 (pour les canaux de couleur) afin de pouvoir les encoder de façon numerique (one hot) par la suite. Par la suite, il sera important de multiplier la matrice qui represente l'image par 255 afin de pouvoir la lire.

(4) **Même question pour les lignes sous le commentaire cf. question 4).** Ici on encode les données en "categorical" pour obtenir les vecteurs des probabilités 0 ou 1 d'appartenance aux différentes classes à partir du vecteur des numéros des classes. On peut ensuite retrouver la prédiction en prenant la plus grande probabilité dans le vecteur de sortie du modèle.

(5) **Rajouter après le commentaire Cf. question 5) le code pour définir le CNN décrit dans la figure (voir sujet TP)** On rajoute le code suivant pour définir l'architecture de notre modèle :

Code Python question 5

```

1  m.add(layers.Conv2D(32, (3, 3), activation="relu"))
2  m.add(layers.Conv2D(32, (3, 3), activation="relu"))
3  m.add(layers.MaxPooling2D((2, 2)))
4  m.add(layers.Flatten())
5  m.add(layers.Dense(128, activation="relu"))
6  # Question 9) config 1bis - dropout en plus dans l'architecture -> accuracy=
   ↪ 0.994999732971191 -> Il y a 95 images mal classees *****MEILLEUR SCORE*****
7  m.add(layers.Dropout(0.25))
8  m.add(layers.Dense(10, activation="softmax"))

```

(6) **Combien de paramètres ce réseau comporte-t-il ?** De manière manuelle nous avons :

- Pour la première couche :  $32 * (3 * 3) + bias_1 = 320$
- Pour la seconde couche :  $32 * (3 * 3) * 32 + bias_2 = 9248$
- Pour la troisième couche : c'est un maxpooling donc pas de paramètres
- Pour la 4ème couche : Flatten aplatis tous les Conv2D en 1 seule dimension, elle n'ajoute pas de paramètres.
- Pour la 5ème couche :  $(4068 * 128) + 128 = 589952$
- Pour la dernière couche : 1290 paramètres
- En tout 600 810 paramètres entraînable.

La fonction `CNN.summary()` retourne en détail le nombre de paramètres.

(7) **Rajoutez après le commentaire Cf. question 7) un code pour évaluer le réseau sur la base de données test. Affichez le pourcentage d'images bien reconnues. Exécutez le programme. Que pensez-vous du résultat obtenu ?** La fonction `evaluate(DT, Dtp)` retourne une liste contenant deux informations : `loss` et `accuracy`. En regardant les données, on observe :

- `loss = 0.07601930946111679`
- `accuracy = 0.991100013256073`

On en déduit que le modèle a prédit 99.11 pourcent des images de la base de test de façon correcte. C'est un très bon résultat.

Le code associé est le suivant :

Code Python question 7

```

1  # Cf. question 7)
2  score = CNN.evaluate(DT, Dt_p)
3  print('accuracy=', score[1])

```

(8) Rajoutez après le commentaire Cf. question 8) un code qui affiche successivement les imagerie qui ont été mal classées, avec le chiffre qui aurait dû être prédit et celui qui a effectivement été prédit ainsi que la probabilité estimée d'appartenance aux différentes classes. Que vous inspirent ces résultats ? Pour cette question nous avons procédé en plusieurs étapes. D'abord nous avons défini une fonction qui permet d'afficher les informations demandées (imagerie, label prédit, label correct et les probabilités de sortie du modèle) :

Code Python question 8 fonction displayimagerie(...)

```

1 def display_imagerie(imagerie, label1, label2, probs):
2     print("Les prédictions pour chaque chiffre sont les suivantes :")
3     for i in range(len(probs)):
4         print(probs[i]) #affichage des probabilités associées à chaque classe
5
6     plt.title('Le chiffre prédit est {label1} , le bon label était
7         ↪ {label2}'.format(label1=label1, label2=label2)) #affichage du label prédit
8         ↪ et du label correct
9
10    image = np.array(imagerie, dtype='float32')*255 #multiplication du tableau par
11    ↪ 255 pour une restitution en niveau de gris
12
13    pixels = image.reshape((28, 28)) # dimensionnement en 28*28 pixels
14
15    plt.imshow(pixels, cmap='gray')
16    plt.show()

```

Une sortie type de cette fonction est :

```

Les prédictions pour chaque chiffre sont les suivantes :
2.149395e-22
6.275622e-17
2.7146692e-12
4.5371356e-09
6.069807e-20
1.1363128e-07
2.725396e-21
4.5944176e-21
0.9999999
4.4195148e-16

Le chiffre prédit est 8 , le bon label était 3
0
5
10
15
20
25
0 5 10 15 20 25

```

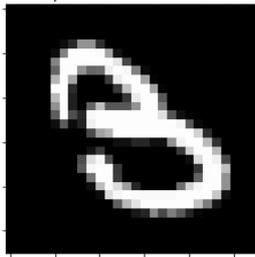


Figure 5: Affichage des probabilités de sortie du modèle, du label prédit, du label correct et de l'imagerie

Après avoir défini cette fonction, nous nous sommes penchés sur l'acquisition des données nécessaires. Pour répondre à la question il fallait récupérer toutes les mauvaises classifications lors du test de notre modèle. Nous avons donc récupéré les images mal classées en comparant la classe prédite d'une image à son label dans les labels de test (fonction `argmax` vue en cours). Si les deux étaient différents, alors l'image était mal classée. Nous avons également récupéré les labels et les probabilités relatives à chaque classe.

Le code suivant illustre de ce processus :

Code Python question 8 récupération des mauvaises classifications

```

1 predictions = (CNN.predict(DT)) #prediction sur les données de test
2 imageries_fausse = [] #tableau qui va contenir les matrices représentant les
3 ↪ imageries.
4 labels = [] #tableau qui va contenir les tuples (labelPrédit, labelCorrect)
5 probs = [] #tableau qui va contenir les probabilités de sortie du modèle
6 for i in range(len(predictions)):

```

```

6     if np.argmax(predictions[i]) != np.argmax(Dt_p[i]): #comparaison de prediction
       ↪ et label
7         imagettes_fausse.append(DT[i])
8         labels.append((np.argmax(predictions[i]), np.argmax(Dt_p[i])))
9         probs.append(predictions[i])
10
11 print("Il y a", len(imagettes_fausse), "images mal classees")

```

Enfin, il etait demande de parcourir les imagerie mal classees via une saisie utilisateur : <Entree> pour continuer et <F> pour arrêter. Le code suivant met en place ce mecanisme :

Code Python question 8 parcours des imageries

```

1 print("Vous allez voir les imagerie mal classees : ")
2
3 for i in range(len(imagettes_fausse)): #on parcourt les imageries mal classees
4     user = input("Appuyez sur <Entree> pour continuer, sur <F> pour arreter") #
       ↪ saisie utilisateur
5
6     if user=="": # equivalent a <Entree>
7
8         #appel de la fonction definie plus haut
9         display_imagerie(imagettes_fausse[i], labels[i][0], labels[i][1], probs[i])
10
11     if user=="F" or user=="f": # sortie du programme
12         print("Vous avez appuye sur <F>, au revoir !")
13         break; #arret de boucle

```

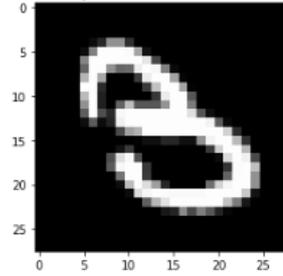
La sortie de ce programme est ici 6.

**(9) Defi : modifiez le code de la fonction def architecture() pour essayer d'obtenir le CNN realisant la prediction la plus performante (en terme de pourcentage d'imagettes de la base de test bien reconnues). Ne modifiez pas le nombre d'epoques defini dans le code minimal donne.** Pour ameliorer le modèle, nous avons ajouter une couche de Dropout après la première couche Dense. Cette couche va permettre de limiter le sur-apprentissage et nous faire gagner quelques centièmes en precision. On passe de 0.991 de precision a 0.994. Le dropout n'ajoute pas de paramètres au modèle nous restons a 600 810. Un autre moyen d'ameliorer le modèle serait d'utiliser le transfer-learning. En effet avec des architectures comme celle du VGG-16, il est possible d'aller jusqu'a 0.999 de precision mais le nombre de paramètres serait superieur a 1 000 000.

```

Vous allez voir les imagerie mal classées :
Appuyez sur <Entrée> pour continuer, sur <F> pour arrêter
Les prédictions pour chaque chiffre sont les suivantes :
2.149395e-22
6.275622e-17
2.7146692e-12
4.5371356e-09
6.069807e-20
1.1363128e-07
2.725396e-21
4.5944176e-21
0.9999999
4.4195148e-16
    
```

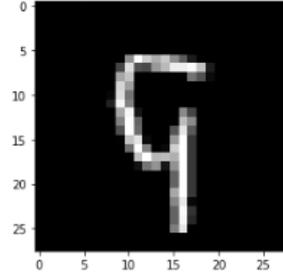
Le chiffre prédit est 8 , le bon label était 3



```

Appuyez sur <Entrée> pour continuer, sur <F> pour arrêter
Les prédictions pour chaque chiffre sont les suivantes :
1.075957e-14
2.9238076e-13
1.3744013e-15
2.2033542e-10
2.91485e-12
0.9416
2.9107803e-14
5.5650233e-08
1.6851498e-08
0.05840003
    
```

Le chiffre prédit est 5 , le bon label était 9



```

Appuyez sur <Entrée> pour continuer, sur <F> pour arrêterf
Vous avez appuyé sur <F>, au revoir !
    
```

Figure 6: Execution du programme, affichage de 2 imagerie mal classees puis sortie du programme

## Code Python Exercice 1

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Feb  1 17:13:13 2022
4
5  @author: Pierre LAGUE - L3 CMI & Luca LEFEVRE L3 STAT
6  """
7
8  import random
9  import numpy as np
10 import statistics
11
12 """
13 1) ecrivez une fonction qui simule toutes les etapes de ce jeu avec en eparamtre la
   ↪ strategie
14 choisie par le candidat et qui renvoie 1 si le joueur gagne, 0 sinon. Utilisez pour
   ↪ les tirages aleatoires la ebibliothqe random et implementez les differents
   ↪ etats du jeu sous la forme de listes
15 de numeros de ibotes (par exemple la liste des deux ibotes restantes une fois le
   ↪ premier choix
16 du candidat effectue). Pour econtrler le deroulement du jeu, affichez les resultats
   ↪ des differentes
17 etapes 'lorsquune variable booleenne globale DEBUG est vraie
18 """
19
20
21 def play(strategy, DEBUG):
22     """
23     Cette fonction simule le jeux des 3 ibotes. Elle utilise des choix
   ↪ pseudo-aleatoires avec la lib random.
24
25     Parameters
26     -----
27     strategy : STRING
28         la strategie du joueur, s1 ou s2, respectivement "le joueur reste sur son
   ↪ premier choix", "le joueur choisi une des deux ibotes restantes".
29     DEBUG : BOOL
30         si c'est vrai, la fonction va afficher les etapes du jeu et les choix du
   ↪ joueur. Sinon, pas d'affichage.
31
32     Returns
33     -----
34     int
35         1 ou 0 respectivement, le joueur a gagne, le joueur a perdu..
36
37     """
38
39     if DEBUG:
40         print("#####")
41         if strategy=="s1":
42             print("Strategy 1 : Le joueur reste sur son premier choix")
43         else:
44             print("Strategy 2 : Le joueur choisi une des deux ibotes restante")
45
46     boxes = [1, 0, 0]
47     boxes = random.sample(boxes, len(boxes))
48     first_move = boxes[random.randint(0, len(boxes)-1)]
49
50
51     if DEBUG:
52         print(boxes)
53         if first_move == 1:
54             print("Le joueur a choisi la ibote contenant le echeque : first_move =",
   ↪ first_move)

```

```

55     else:
56         print("Le joueur a choisi une îbote vide : first_move =", first_move)
57
58     boxes.remove(first_move)
59
60     if DEBUG:
61         print("Les îbotes restantes sont :",boxes)
62         print("#####\n")
63
64     if strategy == "s2":
65         if first_move == 1:
66             second_move = boxes[random.randint(0, 1)]
67         else:
68             second_move = 1
69
70         if DEBUG:
71             if second_move == 1:
72                 print("Le joueur a choisi une des deux îbotes contenant le èchque :
73                     ↪ second_move =", second_move)
74             else:
75                 print("Le joueur a choisi une îbote vide : second_move =",
76                     ↪ second_move)
77                 print("#####\n")
78
79     if strategy=="s1" and first_move == 1:
80         return 1
81     elif strategy=="s2" and second_move == 1:
82         return 1
83     else:
84         return 0
85
86 play("s2", True)
87
88 def play_N_times(n, strategy, DEBUG):
89     """
90     Cette fonction simule N fois le jeu des 3 îbotes en appelant la fonction
91     ↪ play(...)
92
93     Parameters
94     -----
95     n : INT
96         Le nombre de fois que la simulation sera effectuee.
97     strategy : STRING
98         la strategie du joueur, s1 ou s2, respectivement "le joueur reste sur son
99         ↪ premier choix", "le joueur choisi une des deux îbotes restantes".
100     DEBUG : BOOL
101         si c'est vrai, la fonction va afficher les etapes du jeu et les choix du
102         ↪ joueur. Sinon, pas d'affichage..
103
104     Returns
105     -----
106     None.
107
108     """
109     results = []
110     for i in range(n):
111         results.append(play(strategy, DEBUG))
112
113     print("Le nombre de victoire moyen en utilisant la strategie", strategy, "est :
114         ↪ ,np.round(statistics.mean(results), 3), "\n")
115
116 #play_N_times(1000, "s1", True)

```

```
113
114
115 def play_each_strats(n, DEBUG):
116     """
117     Cette fonction simule N fois le jeu des 3 îbotes pour chaque strategie "s1" et
118     ↪ "s2"
119
120     Parameters
121     -----
122     n : int
123         Nombre de fois que la simulation sera effectuee.
124     DEBUG : bool
125         si c'est vrai, la fonction va afficher les etapes du jeu et les choix du
126         ↪ joueur. Sinon, pas d'affichage.
127
128     Returns
129     -----
130     None.
131
132     """
133     play_N_times(n, "s1", DEBUG)
134     play_N_times(n, "s2", DEBUG)
135
136 play_each_strats(5, True)
137 play_each_strats(100000, False)
138 #Converge vers 0.33 pour la strategie 1 et 0.66 pour la strategie 2
```

## Code Python Exercice 2

```

1  """
2  Created on Tue May 17 17:13:13 2022
3
4  @author: Pierre LAGUE - L3 CMI & Luca LEFEVRE L3 STAT
5  """
6  #
7  # Classification 'dimages de chiffres : code minimal
8  #
9
10 # Definition de 'larchitecture du reseau de neurones
11 def def_architecture():
12     m = models.Sequential()
13     # Cf. question 5)
14     m.add(layers.Conv2D(32, (3, 3), activation="relu"))
15     m.add(layers.Conv2D(32, (3, 3), activation="relu"))
16     m.add(layers.MaxPooling2D((2, 2)))
17     m.add(layers.Flatten())
18     m.add(layers.Dense(128, activation="relu"))
19     # Question 9) config 1bis - dropout en plus dans l'architecture -> accuracy=
20     ↪ 0.994999732971191 -> Il y a 95 images mal classees *****MEILLEUR SCORE*****
21     m.add(layers.Dropout(0.25))
22     m.add(layers.Dense(10, activation="softmax"))
23     m.compile(loss='categorical_crossentropy',
24               optimizer='adam',
25               metrics=['accuracy'])
26     return m
27
28 # Apprentissage du reseau. Retourne le modèle.
29 def Apprentissage():
30     m=def_architecture()
31     m.fit(DA, Da_p, epochs=nb_epoques, batch_size=32)
32     return m #2
33
34 # Chargement des bibliothèques et des modules
35 import numpy as np, matplotlib.pyplot as plt
36 from tensorflow.keras import models, layers, utils, backend as K
37 from keras.datasets import mnist
38 from PIL import Image
39
40 # On definit une graine pour des resultats reproductibles
41 np.random.seed(1)
42 # Chargement de la base de donnees MNIST
43 (DA, Da), (DT, Dt) = mnist.load_data()
44
45
46 # Cf. question 2)
47 fig = plt.figure(figsize=(10, 5))
48 question = "Entrez un indice entre 0 et", len(DA)
49 indice = input(question)
50
51 # Plot
52 plt.title('Le chiffre est {label}'.format(label=Da[int(indice)]))
53 plt.imshow(DA[int(indice)], cmap=plt.get_cmap('gray'))
54 plt.show()
55
56 # Cf. question 3) (i)
57 M,N=DA.shape[1], DA.shape[2]
58 if K.image_data_format() == 'channels_first':
59     DA=DA.reshape(DA.shape[0], 1, M, N); DT=DT.reshape(DT.shape[0], 1, M, N)
60     taille = (1, M, N)
61 else:
62     DA=DA.reshape(DA.shape[0], M, N, 1); DT=DT.reshape(DT.shape[0], M, N, 1)

```

```

63     taille = (M, N, 1)
64
65     # Cf. question 3) (ii)
66     #
67     DA=DA.astype('float32'); DA/=255; DT=DT.astype('float32'); DT/=255
68
69     # Cf. question 4)
70     #encodage one-hot
71     Da_p = utils.to_categorical(Da,10); Dt_p = utils.to_categorical(Dt,10)
72     nb_epoques=32
73     CNN=Apprentissage()
74     #print(CNN.summary())
75
76     # Cf. question 7)
77     score = CNN.evaluate(DT, Dt_p)
78     print('accuracy=', score[1])
79
80     def display_imagette(imagette, label1, label2, probs):
81         print("Les predictions pour chaque chiffre sont les suivantes :")
82         for i in range(len(probs)):
83             print(probs[i]) #affichage des probabilites associees a chaque classe
84
85             plt.title('Le chiffre predit est {label1} , le bon label etait
86                 ↪ {label2}'.format(label1=label1, label2=label2)) #affichage du label predit
87                 ↪ et du label correct
88
89             image = np.array(imagette, dtype='float32')*255 #multiplication du tableau par
90                 ↪ 255 pour une restitution en niveau de gris
91
92             pixels = image.reshape((28, 28)) # dimensionnement en 28*28 pixels
93
94             plt.imshow(pixels, cmap='gray')
95             plt.show()
96
97     predictions = (CNN.predict(DT)) #prediction sur les donnees de test
98     imagettes_fausse = [] #tableau qui va contenir les matrices representant les
99     ↪ imagettes.
100     labels = [] #tableau qui va contenir les tuples (labelPredit, labelCorrect)
101     probs = [] #tableau qui va contenir les probabilites de sortie du modèle
102     for i in range(len(predictions)):
103         if np.argmax(predictions[i]) != np.argmax(Dt_p[i]): #comparaison de prediction
104             ↪ et label
105             imagettes_fausse.append(DT[i])
106             labels.append((np.argmax(predictions[i]), np.argmax(Dt_p[i])))
107             probs.append(predictions[i])
108
109     print("Il y a", len(imagettes_fausse), "images mal classees\n")
110
111     print("Vous allez voir les imagette mal classees : \n")
112
113     for i in range(len(imagettes_fausse)): #on parcourt les imagettes mal classifiees
114         user = input("Appuyez sur <Entree> pour continuer, sur <F> pour éarrter") #
115             ↪ saisie utilisateur
116
117         if user=="": # equivalent a <Entree>
118             #appel de la fonction defnime plus haut
119             display_imagette(imagettes_fausse[i], labels[i][0], labels[i][1], probs[i])
120
121         if user=="F" or user=="f": # sortie du programme
122             print("Vous avez appuye sur <F>, au revoir !")
123             break; #arret de boucle

```