

Correction du DS 3

Julien REICHERT

Partie 1

Exercice 1.1

Une dimension par pixel de l'image, soit $28 \times 28 = 784$ pour la valeur de d .

Exercice 1.2

```
def difference(im1, im2):
    rep = 0
    for i in range(len(im1)): # 28
        for j in range(len(im1[i])): # 28 partout
            if im1[i][j] != im2[i][j]:
                rep += 1
    return rep
```

Exercice 1.3

```
def kNN(images, image_sup, k):
    distances = [(difference(image, image_sup), image[1]) for image in images]
    distances.sort()
    etiquettes = [0] * 10
    for _, etiq in distances[:k]:
        etiquettes[etiq] += 1
    rep = 0
    for i in range(1, 9):
        if etiquettes[i] > etiquettes[rep]:
            rep = i
    return rep
```

Exercice 1.4

Les variables sont ici considérées comme globales.

```
def efficacite(k):
    resultats = [[0] * 10 for i in range(10)] # Ligne i : c'est un i, colonne j : l'algorithme dit j
    for (image_test, chiffre) in TEST:
        etiquette = kNN(ENTRAINEMENT, image_test, k)
        resultats[chiffre][etiquette] += 1
    reussites = 0
    for i in range(10):
        reussites += resultats[i][i]
    return reussites / len(TEST) * 100
```

Exercice 1.5

On considère que le tri effectué par la méthode `sort` est optimisé pour avoir une complexité en $\mathcal{O}(n \log n)$. Pour $n = 70000$, cela fait donc environ 70000×16 donc 1120000 comparaisons et en termes d'opérations élémentaires on peut compter ceci en double, soit à la louche 25000000 ns = 0,025 s pour le tri à chaque fois. Ceci s'effectue pour les 10000 images, donc on a un total de 250 s.

La comparaison de deux images prenant une microseconde, la comparaison d'une image de test à toutes les images d'entraînement en prend 70000 soit 0,07 s. Ceci s'effectue pour les 10000 images, donc on a un total de 700 s.

Les accès au dictionnaire en vue de l'étiquetage sont au nombre de 70000 et le temps total sera négligé (total inférieur à une milliseconde pour chaque donnée de test).

On peut alors estimer le temps total à entre 1 s et 1000 s, mais vu la proximité du résultat calculé à 1000 s, on tolère aussi la réponse « entre 1000 s et 1000000 s », le respect de la consigne étant essentiel.

Après vérification sur internet, les durées estimées sont entre dix et vingt minutes, rendant le calcul à la louche très pertinent !

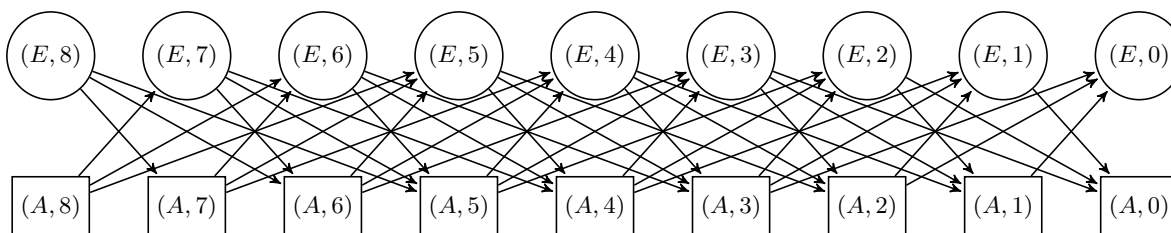
Partie 2

Exercice 2.1

Les configurations gagnantes pour Ève sont les configurations $(\text{Ève}, k)$ pour k non multiple de 4 entre 0 et 21 et les configurations (Adam, k) pour k multiple de 4 entre 0 et 21.

Exercice 2.2

Par flemme, l'essentiel du travail a été laissé à ChatGPT. Puis il a fallu corriger...



Exercice 2.3

Plutôt que de refaire un dessin, on décrit les ensembles ici :

- $\mathcal{A}_0 : \{(A, 0)\}$
- $\mathcal{A}_1 \setminus \mathcal{A}_0 : \{(E, 1); (E, 2); (E, 3)\}$
- $\mathcal{A}_2 \setminus \mathcal{A}_1 : \{(A, 4)\}$
- $\mathcal{A}_3 \setminus \mathcal{A}_2 : \{(E, 5); (E, 6); (E, 7)\}$
- $\mathcal{A}_4 \setminus \mathcal{A}_3 : \{(A, 8)\}$

Exercice 2.4

```
def initial():  
    return [[], [1], [1, 2]] + [[1, 2, 3] for i in range(3, 22)]
```

Exercice 2.5

```
from random import choice

def maj_partie(coups_Eve, coups_Adam):
    batonnets = 21
    joueur = 0
    coups = [coups_Eve, coups_Adam]
    partie = []
    while True:
        liste = coups[joueur][batonnets]
        if liste == []: # Abandon
            if len(partie) >= 2: # Sinon, abandon immédiat d'Adam avec 20 bâtonnets
                dernier = batonnets + partie[-1] + partie[-2]
                coups[joueur][dernier].remove(partie[-2]) # suppression dernier coup
            return
        else:
            coup = choice(liste)
            partie.append(coup)
            batonnets -= coup
            joueur = 1 - joueur
```

Exercice 2.6

Petite remarque la condition d'arrêt suggérée par l'énoncé est nécessaire parce que si elle est vérifiée la fonction `maj_partie` n'a pas d'effet et on risque de continuer à l'infini.

« Lancer l'expérience » revient à initialiser, ce qui est cohérent avec la consigne de ne pas mettre d'argument.

```
def experience():
    coups_Eve = initial()
    coups_Adam = initial()
    while coups_Adam[20] != []:
        maj_partie(coups_Eve, coups_Adam)
    return coups_Eve, coups_Adam
```

Exercice 2.7

Le seul coup gagnant d'Ève étant 1 en début de partie, c'est le seul qui ne peut pas être supprimé. Ceci étant, si Ève joue systématiquement optimalement (possible avec certes une probabilité infime), les listes aux indices multiples de quatre d'Adam finiront par se vider et l'expérience s'arrêtera sans qu'Ève ne retire le moindre coup de sa liste.

Exercice 2.8

Dans le pire des cas, il aura fallu découvrir que tous les coups perdants accessibles le sont, donc pour chaque configuration multiple de quatre (sauf 20 bâtonnets quand c'est au tour d'Ève qui n'est pas accessible et les configurations finales correspondant déjà à des listes vides) il faut trois parties, pour un total de 24, et pour chaque autre configuration, il en faut deux (sauf s'il reste deux bâtonnets où il en faut une car il n'y a que deux choix de base), soit $27 \times 2 + 2 \times 1 = 56$ parties de plus, pour un total final de 80 parties.

Partie 3

Exercice 3.1

```
def scores(resultats):
    n = len(resultats)
    assert n > 0
    dix = len(resultats[0][1])
    scores = [0] * dix
    for reponse, propositions in resultats: # calcul des plus proches en mode bourrin
        gameover = abs(reponse) + max([abs(p) for p in propositions]) + 1 # Pire que la pire réponse !
        prop = propositions[:] # Copie qu'on va détruire
        optima = []
        while len(optima) < 3:
            meilleurs = [0]
            distmin = abs(prop[0] - reponse)
            for i in range(1, len(prop)):
                dist = abs(prop[i] - reponse)
                if dist < distmin:
                    meilleurs = [i]
                    distmin = dist
                elif dist == distmin:
                    meilleurs.append(i)
            for indice in meilleurs:
                scores[indice] += 3 - len(optima)
                prop[indice] = gameover
            optima.extend(meilleurs)
    return scores
```

Exercice 3.2

Tri fusion pour la rapidité (certes, on est sur des listes de taille dix donc l'intérêt est limité...)

```
def merge(l1, l2):
    n1, n2, i1, i2 = len(l1), len(l2), 0, 0
    l = [None] * (n1+n2)
    while i1 < n1 and i2 < n2:
        if l1[i1][1] > l2[i2][1]:
            l[i1+i2] = l1[i1]
            i1 += 1
        else:
            l[i1+i2] = l2[i2]
            i2 += 1
    for i in range(i1, n1): # vide si i1 = n1, termine l
        l[i+i2] = l1[i]
    for i in range(i2, n2): # vide si i2 = n2, termine l
        l[i1+i] = l2[i]
    return l

def merge_sort(l):
    n = len(l)
    if n <= 1:
        return l
    return merge(merge_sort(l[:n//2]), merge_sort(l[n//2:]))
```

```

def classement(liste):
    liste_avec_indice = [(i, liste[i]) for i in range(len(liste))]
    ordre = merge_sort(liste_avec_indice)
    resultat = []
    place = 1
    valeur_precedente = None
    for i in range(len(ordre)):
        (nom, valeur) = ordre[i]
        if valeur_precedente != valeur:
            place = i + 1 # On réinitialise la place qui ne peut être que l'indice actuel plus un
            valeur_precedente = valeur
        resultat.append((nom, place))
    return resultat

```

Exercice 3.3

```

SELECT reponse FROM SOUMISSIONS WHERE equipe = -1 AND question = 8

```

Exercice 3.4

```

SELECT ABS(S.reponse - BR.reponse)
FROM SOUMISSIONS AS S
JOIN SOUMISSIONS AS BR
ON S.question = BR.question
WHERE S.equipe = 5 AND BR.equipe = -1 AND S.question = 1

```

Exercice 3.5

```

SELECT S.question, ABS(S.reponse - BR.reponse)
FROM SOUMISSIONS AS S
JOIN SOUMISSIONS AS BR
ON S.question = BR.question
WHERE S.equipe = 0 AND BR.equipe = -1

```

Exercice 3.6

```

SELECT ABS(S.reponse - BR.reponse) AS distance
FROM SOUMISSIONS AS S
JOIN SOUMISSIONS AS BR
ON S.question = BR.question
WHERE S.equipe <> -1 AND BR.equipe = -1 AND S.question = 2
ORDER BY distance LIMIT 1

```

Exercice 3.7

```

SELECT S.equipe, COUNT(*)
FROM SOUMISSIONS AS S
JOIN SOUMISSIONS AS BR
ON S.question = BR.question
WHERE S.equipe <> -1 AND BR.equipe = -1 AND S.reponse = BR.reponse
GROUP BY S.equipe

```

Exercice 3.8

```
SELECT equ, AVG(distances) AS moyenne FROM
(
  SELECT S.equipe AS equ, ABS(S.reponse - BR.reponse) AS distance
  FROM SOUMISSIONS AS S
  JOIN SOUMISSIONS AS BR
  ON S.question = BR.question
  WHERE S.equipe <> -1 AND BR.equipe = -1
)
GROUP BY S.equipe
ORDER BY moyenne
```