# CONSIGNES GÉNÉRALES

Document non autorisés
L'usage des calculatrices est interdit

Ce cahier d'examen comporte 30 pages + 2 pages supplémentaires
Les réponses doivent être écrites dans
Les espaces de réponses dédiés
En cas de besoin utiliser les pages vides en fin du cahier en le signalant dans l'espace de réponse correspondant

Il est impératif de respecter la nomenclature des objets de l'énoncé
Utiliser éventuellement l'annexe

Les énoncés et les modélisations présentés dans les deux parties ont été adaptés spécifiquement pour les besoins de l'épreuve pour des fins d'apprentissage et d'évaluation uniquement.

Le sujet comporte deux parties qui traitent les aspects suivants :

**Partie I** : Programmation orientée objet (**Q1**.. **Q20**).

**Partie II** : Base de données relationnelle (**Q21**..**Q32**).

# Partie I : Programmation Python

Le modèle **Skip-Gram** est une technique clé en **traitement automatique du langage naturel (NLP)**, conçue pour apprendre des représentations vectorielles des mots en fonction de leur contexte d'utilisation. Cette approche, souvent désignée par le terme de **Word Embedding**, permet de capturer les relations entre les mots de manière efficace, améliorant ainsi les performances de diverses applications NLP telles que les chatbots, la traduction automatique, etc.

Les étapes clés pour construire et entraîner un modèle de réseau de neurones artificiel appelé Skip-Gram destiné à l'apprentissage des représentations vectorielles des mots (**Word Embedding**) sont les suivantes :

1. **Prétraitement des données :** Charger les données textuelles, effectuer un nettoyage (filtrage des caractères spéciaux, etc.), et convertir les mots en représentations numériques (indexation du vocabulaire, création des paires **cible-contexte**).
2. **Préparation des éléments du modèle :** Définir les blocs de construction d'un réseau de neurones. Notamment les couches **dense** et d'**embedding**.
3. **Edification, entraînement et évaluation du modèle :** Assembler les éléments établis dans l'étape précédente pour définir un réseau de neurone séquentiel quelconque.
4. **Simulation du modèle Skip-Gram :** Instancier un réseau de neurone selon l'architecture du modèle Skip-Gram ensuite l'entraîner sur les paires cible-contexte générées à l'étape de prétraitement et évaluer ses performances en mesurant la qualité des embeddings générés sur un ensemble de données de test.

## Classes à Implémenter

Tout au long de ce sujet, nous aurons à implémenter les classes suivantes :

— `Tokenizer` : Une classe qui permet d'effectuer la première étape de prétraitement des données, ses attributs représentent les critères de filtrage ainsi que le vocabulaire résultant. L'implémentation de cette classe fera objet de l'étape 1.

— `Layer` : C'est une classe abstraite (dédiée à l'héritage) elle représente l'interface commune d'une couche quelconque d'un réseau de neurones. Ses attributs incluent la taille des entrées (`input_size`), la taille des sorties (`output_size`) et les poids des connexions entre les neurones (`w`). Elle comporte aussi l'entrée actuelle (`x`) et la sortie actuelle (`y`).

— `Embedding` : Une classe dérivée de `Layer`, ses poids (`w`) capturent la représentation vectorielle des mots ; sa dimension d'entrée (`input_size`) est égale à la taille du vocabulaire et sa dimension de sortie (`output_size`) est la taille de la représentation vectorielle apprise.

— `Dense` : Une classe fille de `Layer` qui applique une opération linéaire sur sa matrice d'entrée (`x`) suivie d'une fonction non-linéaire (**`softmax`**). L'implémentation des trois classes `Layer`, `Embedding` et `Dense` fera objet de l'étape 2.

— `NeuralNetwork` : Une classe qui représente une séquence de couches (instances de sous classes de la classe `Layer`). Elle est responsable du processus d'entrainement du modèle (apprentissage) qui consiste à calculer (prédire) la sortie finale, de mesurer la perte par rapport aux données réelles pour ajuster les poids de ses différentes couches. Ce processus est repris durant plusieurs **époques** afin de rapprocher sa sortie finale (la prédiction) aux sorties attendues (réalités terrain). L'implémentation de cette classe fera objet de l'étape 3.

## Etape I : Prétraitement des données

Cette étape consiste à préparer les données textuelles en les nettoyant et en les convertissant sous une forme numérique adaptée au modèle. Voici un processus détaillé de préparation des données pour le modèle Skip-gram :

1. **Nettoyage et Tokenisation du texte :** Filtrage des caractères spéciaux, des chiffres inutiles et découpage du texte en mots (tokens).
2. **Création du vocabulaire :** Un vocabulaire est construit en collectant tous les mots uniques dans le corpus. Cela permet de créer un index pour chaque token.
3. **Conversion en index :** Remplacer les mots par leurs indices dans le vocabulaire obtenu à l'étape précédente. Ces index formeront (x) l'entrée du réseau de neurones.
4. **Création de paires mot-contexte :** Pour chaque token dans le corpus, des paires de la forme (token, token-contexte) sont créées en associant le mot avec des mots avoisinant (autrement dit le voisinage d'un token donné définit sa **fenêtre contextuelle**).

## Interface de la classe `Tokenizer`

### Rôle & responsabilités

La classe `Tokenizer` permet de découper des textes en tokens puis les indicer.

### Attributs

- ♠ `filters` : Une chaine contenant les caractères à filtrer (nettoyer) du texte en entrée, comme la ponctuation et les symboles spéciaux. Valeur par défaut :

```
'!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n'
```

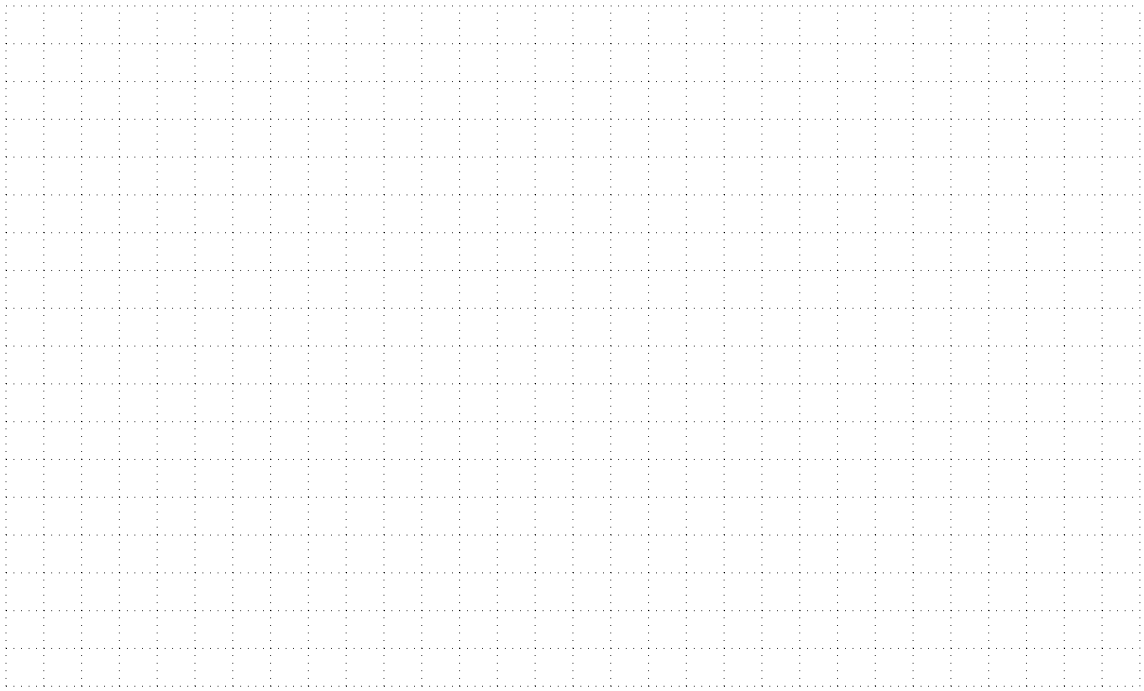
- ♠ `lower` : Un booléen indiquant si le texte en entrée doit être converti en minuscules avant le traitement. Valeur par défaut `True`.
- ♠ `oov_token` : Une Chaîne de caractères à utiliser pour les mots hors vocabulaire (rencontrés après la phase d'apprentissage). Lorsqu'un mot n'est pas présent dans le vocabulaire appris, il est remplacé par `oov_token`. Valeur par défaut `None`.
- ♠ `word_index` : Un dictionnaire qui mappe chaque mot unique à un entier. Chaque clé est un mot unique présent dans les textes. Chaque valeur est un entier unique associé au mot. L'indice commence à 1 (0 est réservé pour les mots hors vocabulaire (`oov_token`)) et les indices sont attribués dans l'ordre d'apparition des mots lors de l'analyse des textes. Ce dictionnaire représente le vocabulaire appris durant la phase des pré-traitements.
- ♠ `split` : Une chaîne de caractères utilisée pour diviser le texte en tokens. Valeur par défaut le caractère espace.

### Méthodes à implémenter

**Q1.** `__init__(...)` : Initialise une nouvelle instance à partir des paramètres `filters`, `lower`, `oov_token` et `split`. L'attribut `word_index` est initialisé par un dictionnaire singleton où la clé `oov_token` est associée à l'indice 0.

### Espace de réponse pour Q1

```
import numpy as np
class Tokenizer :
   def __init__(self,
                filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
                lower=True, oov_token=None, split=" "):
```



**Q2**. `tokenize(...)` : Prend en paramètre la liste de textes `texts` et permet de prétraiter chaque texte de `texts` en appliquant les actions suivantes :

- ♠ Convertir le texte en minuscule, si l'attribut `lower` est défini comme `True`.
- ♠ Remplacer chaque caractère du texte trouvé dans l'attribut `filters` par le caractère spécifié dans l'attribut `split`.
- ♠ Découper le texte nettoyé en une liste de mots en utilisant l'attribut `split` comme séparateur. Ces mots seront appelés des **tokens**.

La méthode retourne une liste de listes, où chaque sous-liste contient les mots d'un texte.

### Illustration sur exemple

```
texts=[ "Installer un logiciel de développement", "un@web_developer.com"]

tk = Tokenizer()
words = tk.tokenize(texts)
print(words)
[
 ['installer', 'un', 'logiciel', 'de', 'développement'],
 ['un', 'web', 'developer', 'com']
]
```

### Espace de réponse pour Q2

```
def tokenize(self, texts):
```

**Q3**. `fit_on_texts(...)` : Cette méthode prend en entrée `texts` une liste de textes et met à jour le dictionnaire `word_index` (représentant le vocabulaire) en fonction de ces textes.

### Suite illustration

```
tk.fit_on_texts(texts)
print(tk.word_index)
{None: 0, 'installer': 1, 'un': 2, 'logiciel': 3,
'de': 4, 'développement': 5, 'web': 6, 'developer': 7, 'com': 8}
```

### Espace de réponse pour Q3

```
def fit_on_texts(self, texts):
    index = len(self.word_index)
```

**Q4**. `to_sequences(...)` : Accepte une liste de textes `texts` et retourne une liste de listes d'entiers où chaque entier représente la valeur d'un token de texte de `texts` .

### Suite illustration

```
print(tk.to_sequences(texts))
[[1, 2, 3, 4, 5], [2, 6, 7, 8]]
```

### Espace de réponse pour Q4

```
def to_sequences(self, texts):
    words = self.tokenize(texts)
```

**Q5**. `generate_pairs(...)` : Génère, à partir d'une liste de textes `texts` et d'une taille de fenêtre contextuelle `window_size` (valeur par défaut=2), une liste de tuples. La méthode parcourt `texts`, exploite les indices des tokens dans le dictionnaire `word_index` pour créer des tuples `(idx_token_cible, idx_token_contexte)`. Pour la constitution de chaque tuple, on considère l'indice d'un token cible et l'indice d'un de ses tokens voisins dans la fenêtre contextuelle. La figue 1 illustre un exemple de fenêtre contextuelle d'un token `(window_size = 4)`.

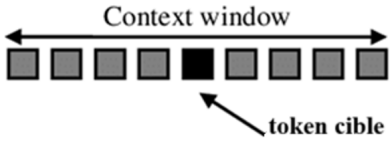



FIGURE 1 – Illustration de fenêtre contextuelle.

Dans la figure 2, pour le token cible `"logiciel"` et pour un (`window_size = 2`) nous obtenons le contexte : `["installer", "un", "de", "développement"]`.

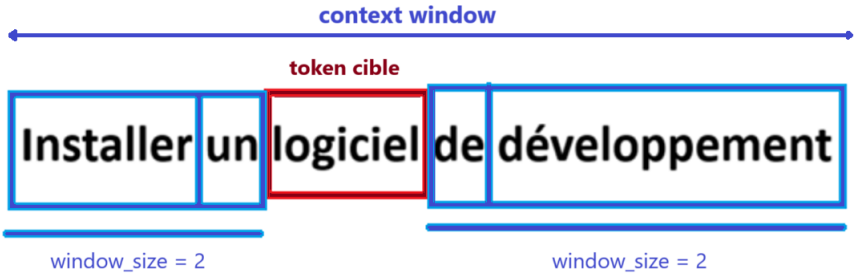



FIGURE 2 – Taille d'une fenêtre contextuelle.

Pour le cas du texte `"Configurer un serveur web"` avec `window_size=2`, si le mot cible est `"serveur"`, les mots de contexte sont : `["Configurer", "un", "web"]`.

### Suite illustration

L'illustration montre le résultat de traitement de `generate_pairs` pour l'exemple de la page 4.

```
print(tk.generate_pairs(texts))
[
(1, 2), (1, 3), (2, 1), (2, 3), (2, 4), (3, 1), (3, 2), (3, 4), (3, 5),
(4, 2), (4, 3), (4, 5), (5, 3), (5, 4), (2, 6), (2, 7), (6, 2), (6, 7),
(6, 8), (7, 2), (7, 6), (7, 8), (8, 6), (8, 7)
]
```

### Espace de réponse pour Q5

```
def generate_pairs (self, texts, window_size=2):
```

**Q6**. `generate_data(...)` : Génère, à partir d'une liste de textes `texts` et d'une taille de fenêtre contextuelle `window_size` (valeur par défaut=2), un tuple `(x,y_true)` où :

- `x` est un vecteur (de taille égale au nombre de paires générées par la méthode `generate_pairs` sur `texts`) ce vecteur contient les indices des tokens cible de chaque paire.
- `y_true` est une matrice ayant comme shape : (nombre de paires, taille du vocabulaire) = `(x.size, len(word_index))`.

**Étapes à suivre**

1. Générer `context_indices` la liste des paires à partir de `texts` en appelant la méthode `generate_pairs`.
2. Générer la liste `lx` contenant les indices des tokens cibles.
3. Générer la liste `ly` contenant les indices des tokens contexte.
4. Convertir la liste `lx` en un vecteur `x`.

5. Créer une matrice nulle `y_true` de shape `(len(ly), vocab_size)` pour stocker les vecteurs qui encodent les tokens contextuels. Les lignes de cette matrice sont représentées selon l'encodage one-hot où toutes les entrées de chaque ligne sont nulles exceptée l'entrée correspondante à l'indice du token contexte ayant la valeur 1. Pour chaque index `i` de `ly`, on assigne la valeur 1 à l'élément correspondant de `y_true`, situé à la position `(i,ly[i])`.

### Suite illustration

L'illustration montre un extrait de traitement de `generate_data` pour l'exemple de la page 4.

```
texts = ["web_developer.de"]
tk.to_sequences(texts) # donne
[[6, 7, 4]]
tk.generate_pairs(texts) # donne
[(6, 7), (6, 4), (7, 6), (7, 4), (4, 6), (4, 7)]
tk.generate_data(texts) # donne
(array([6, 6, 7, 7, 4, 4]),
 array([[0., 0., 0., 0., 0., 0., 0., 1., 0.],
        [0., 0., 0., 0., 1., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 1., 0., 0.],
        [0., 0., 0., 0., 1., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 1., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 1., 0.]]))
```

### Espace de réponse pour Q6

```
def generate_data (self, texts, window_size=2):
    context_indices = self.generate_pairs(texts, window_size)
```

## Etape II : Préparation des éléments du modèle

Un réseau de neurones est formé par la juxtaposition de couches communicantes pour enchainer les différentes étapes du modèle. `Layer`, `Embedding` et `Dense` sont les classes à implémenter durant cette étape représentent les blocs élémentaires formant un réseau de neurones.

## Interface de la classe `Layer`

### Rôle & responsabilités

La classe `Layer`, dont les autres couches hériteront, contient les caractéristiques communes à toutes les couches.

### Attributs

- ♠ `input_size` : un entier qui représente le nombre de neurones en entrée de la couche précédente.
- ♠ `output_size` : un entier qui représente le nombre de neurones dans la couche.
- ♠ `w` : matrice des poids de la couche de shape (`input_size`, `output_size`).
- ♠ `x` : l'entrée de la couche, initialisée par `None`.
- ♠ `y` : la sortie de la couche, initialisée par `None`.

### Méthodes à implémenter

**Q7**. `__init__(...)` : Initialise les attributs d'une couche à partir des paramètres `input_size`, `output_size`. La matrice `w` est initialisée par des valeurs aléatoires flottantes comprises entre $-a$ et $a$ tel que :

$$a = \sqrt{\frac{6}{input\_size + output\_size}} \tag{1}$$

---

**Recommandation :** Utiliser la fonction `np.random.uniform(low, high, size = (l, c))` qui retourne une matrice de shape `(l, c)` remplie par des valeurs flottantes aléatoires selon la loi uniforme comprises entre `low` et `high`.

---

### Espace de réponse pour Q7

```
class Layer:
    def __init__(self,input_size,output_size):
```

La méthode `forward(...)` : Calcule la sortie `y` d'une couche pour un `x` donné, `x` est un vecteur ou une matrice où la première dimension correspond au nombre de valeurs d'entrées (`batch_size`).

$$X \rightarrow \boxed{\text{layer}} \rightarrow Y$$

FIGURE 3 – Illustration de la propagation(avant) dans une couche.

```python
def forward (self, x):
        # pour cette classe le script de cette méthode n'est pas demandé
        pass
```

La méthode `backward(...)` : Calcule $\frac{dE}{dX}$ pour un $\frac{dE}{dY}$ donné (et met à jour les paramètres)

$$\frac{\partial E}{\partial X} \leftarrow \boxed{\text{layer}} \leftarrow \frac{\partial E}{\partial Y}$$

FIGURE 4 – Illustration de la rétro-propagation (arrière) dans une couche.

```python
def backward(self, e, learning_rate):
   # pour cette classe le script de cette méthode n'est pas demandé
   pass
```

## Interface de la classe fille `Embedding`

### Rôle & responsabilités

La couche `Embedding` est responsable de projeter les indices de tokens d'un vocabulaire dans un espace vectoriel continu, où les tokens avec des significations similaires sont proches les uns des autres. Cette couche prend en entrée un vecteur (`x`) contenant les indices des tokens du vocabulaire et renvoie leurs embeddings correspondants.

**Q8**. `forward`(...) : Prend en entrée un vecteur d'indices `x` de shape (`batch_size`,), assigne sa valeur à l'attribut `x`, puis calcule, met à jour et retourne la valeur de l'attribut `y` (une matrice de shape (`batch_size`, `output_size`)). Chaque ligne $y_i$ de `y` reçoit la ligne $w_{x_i}$.

### Illustration sur exemple

$$w = \begin{bmatrix} 0.123 & 0.456 & 0.789 \\ 0.234 & 0.567 & 0.890 \\ 0.345 & 0.678 & 0.901 \\ 0.456 & 0.789 & 0.012 \\ 0.567 & 0.890 & 0.123 \end{bmatrix}$$

Pour $x = \begin{bmatrix} 0 & 3 & 1 & 3 \end{bmatrix}$

On obtient

$$y = \begin{bmatrix} 0.123 & 0.456 & 0.789 \\ 0.456 & 0.789 & 0.012 \\ 0.234 & 0.567 & 0.890 \\ 0.456 & 0.789 & 0.012 \end{bmatrix}$$

### Espace de réponse pour Q8

```
class Embedding(Layer):
    def forward(self,x):
```

**Q9**. `backward(...)` : Cette méthode dédiée à la phase de rétro-propagation, met à jour les poids de la matrice `w` afin de minimiser l'erreur du modèle. Cet ajustement exploite le gradient de l'erreur par rapport aux composantes de `y` (matrice de dérivées partielles $\frac{de}{dy}$). Cette méthode retourne le gradient de l'erreur par rapport à `x` (dénotée par $dx$) pour la couche précédente (une matrice de shape (`batch_size` , `output_size`)).

**Paramètres d'entrée :**

1. Une matrice `e` de shape (`batch_size` , `output_size`) représentant le gradient de la perte par rapport à l'embedding du mot cible.
2. Un flottant `learning_rate` qui correspond au taux d'apprentissage, déterminant la magnitude de la mise à jour appliquée aux poids.

**Mise à jour de `w` et Calcul de `dx` :**

La matrice d'embedding `w` est modifiée selon la formule suivante : Pour chaque élément `token_index` d'indice `i` dans `x` :

$$W_{\text{token\_index}} = W_{\text{token\_index}} - \text{learning\_rate} * e_i \tag{2}$$

$$dx_i = e_i \text{ (composantes du vecteur à retourner)} \tag{3}$$

### Espace de réponse pour Q9

```
def backward(self,e,learning_rate):
```

## Interface de la classe fille `Dense`

### Rôle & responsabilités

Cette classe représente une couche où tous les neurones d'entrée sont connectés à tous les neurones de sortie. Les poids ($w_{ij}$) reflètent l'intensité de connectivité entre un neurone $i$ (en entrée) et un neurone $j$ (en sortie), voir figure 5.

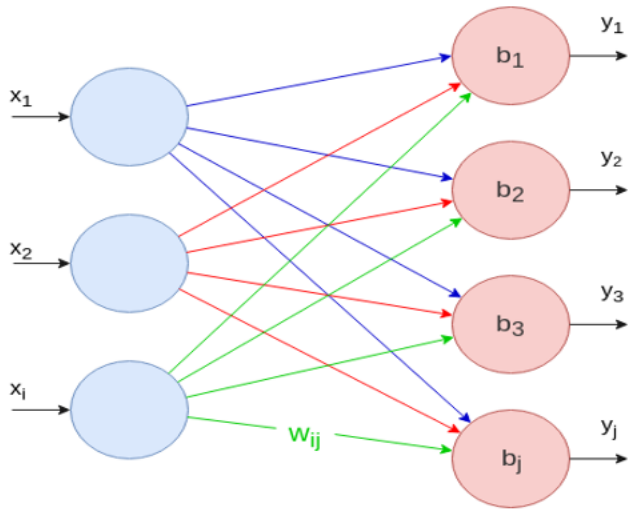



FIGURE 5 – Illustration d'une couche dense.

### Attributs

- `b` : Biais de la couche, cet attribut est initialisé avec un vecteur nul de taille (`output_size`) et mis à jour lors de l'apprentissage du modèle.

### Méthodes à implémenter

**Q10**. `__init__(...)` : Initialise une couche dense à partir des paramètres `input_size`, `output_size`.

#### Espace de réponse pour Q10

```
class Dense(Layer):
    def __init__(self, input_size, output_size):
```

**Q11**. `forward(...)` : Assigne à l'attribut `x` le paramètre `x` (une matrice de shape (`batch_size`, `input_size`)), calcule et retourne la sortie `y` (une matrice de shape (`batch_size`, `output_size`)) de la couche `Dense` en procédant comme suit :

1. Effectue une multiplication matricielle entre la matrice `x` et les poids `w` et ajoute le biais :

$$z = x\ w + b \tag{4}$$

2. Applique la fonction d'activation pour obtenir la sortie finale

$$y = \texttt{softmax}(z) \tag{5}$$

La fonction `softmax` définie comme suit :

$$\texttt{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{size(z)} e^{z_j}} \tag{6}$$

Cette fonction est utilisée pour obtenir une distribution de probabilité sur les classes.

#### Espace de réponse pour Q11

```
    def forward(self, x):
```

**Q12**. `backward(...)` : Prend en entrée la matrices `e` et le flottant `learning_rate` qui correspondent respectivement aux gradient de perte et le taux d'apprentissage. Cette méthode calcule les gradients relativement aux poids et aux biais, ajuste les poids (`w`) et les biais (`b`) suivant les gradients obtenus puis calcule et retourne le gradient de l'erreur par rapport à (`x`) pour la rétropropagation ($\frac{\partial e}{dx}$). Voici les étapes détaillées :

1. **Calculer le gradient d'activation ($\frac{\partial e}{dz}$)**

$$\frac{\partial e}{dz} = e \text{ (par simplification)} \tag{7}$$

2. **Calculer le gradient relativement aux poids ($\frac{\partial e}{dw}$)**

$$\frac{\partial e}{dw} = x^T \frac{\partial e}{dz} \tag{8}$$

   $x^T$ est la transposée de l'entrée $x$

3. **Calculer le gradient relativement aux biais $\frac{\partial e}{db}$**

$$\frac{\partial e}{db_j} = \sum_{i=0}^{batch\_size-1} \frac{\partial e}{dz_{ij}} \tag{9}$$

   Ainsi, $\frac{\partial e}{db}$ est le vecteur contenant la somme des colonnes de la matrice `e`.

4. **Mise à jour des poids et biais** Les poids et les biais sont ajustés à l'aide de la règle de descente de gradient :

$$w = w - \alpha \frac{\partial e}{dw} \tag{10}$$

$$b = b - \alpha \frac{\partial e}{db} \tag{11}$$

   Où $\alpha$ est le taux d'apprentissage (`learning_rate`).

5. **Gradient des entrées ($\frac{\partial e}{dx}$)** Ce gradient est nécessaire pour transmettre l'erreur à la couche précédente :

$$\frac{\partial e}{dx} = \frac{\partial e}{dz} w^T \tag{12}$$

### Espace de réponse pour Q12

```
def backward(self,e,learning_rate):
```

## Étape III : Édification, entraînement et évaluation du modèle

La construction d'un réseau de neurones profond consiste à séquencer une suite de couches. En l'occurrence le modèle Skip-Gram objet de notre simulation est édifié sur la base d'une couche `Embedding` suivie d'une couche `Dense`. Dans la suite nous allons implémenter la classe `NeuralNetwork` qui représente un réseau de neurone séquentiel quelconque.

### Interface de la classe `NeuralNetwork`

#### Rôle & responsabilités

Cette classe représente un réseau de neurone séquentiel quelconque son interface permet de déclencher la propagation en avant (`forward`) et en arrière (`backward`) à travers ses couches. Elle offre aussi des moyens pour évaluer la performance et mesurer l'erreur de prédiction relativement à une vérité terrain.

#### Attributs

- ♠ `layers` : Liste des couches du réseau.
- ♠ `learning_rate` : Un flottant qui correspond au taux d'apprentissage, valeur par défaut=0.01.

#### Méthodes à implémenter

**Q13**. `__init__(...)` : Initialise l'attribut `layers` par une liste vide pour stocker les couches du réseau, et initialise l'attribut `learning_rate` avec la valeur fournie en paramètres.

### Espace de réponse pour Q13

```
class NeuralNetwork:
    def __init__(self ,learning_rate = 1e-2):
```

**Q14**. `add_layer(...)` : Permet d'ajouter une couche `l` au réseau.

### Espace de réponse pour Q14

```
    def add_layer(self,l):
```

**Q15**. `forward`(...) : Effectue la propagation des données d'entrée `x` (un vecteur de taille `batch_size` contenant les données d'entrée ) à travers toutes les couches du réseau. Cette méthode parcourt chaque couche `l` dans la liste des couches du réseau et applique la méthode `forward` de la couche à l'entrée actuelle `x`. La sortie de chaque couche devient automatiquement l'entrée pour la couche suivante, jusqu'à obtenir la sortie finale du réseau (voir figure 6).

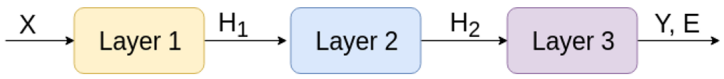



FIGURE 6 – Illustration de la propagation avant dans un réseau de neurones.

### Espace de réponse pour Q15

```
    def forward(self,x):
```

**Q16**. `backward(...)` : Calcule les gradients pour toutes les couches du réseau en effectuant une rétro-propagation de l'erreur. La méthode parcourt les couches du réseau dans l'ordre inverse et utilise la méthode `backward` de chaque couche pour propager le gradient d'erreur. Elle prend comme entrée `e` le gradient d'erreur par rapport à la sortie du modèle $\frac{\partial e}{dy}$ et retourne le gradient d'erreur par rapport à l'entrée du modèle $\frac{\partial e}{dx}$.

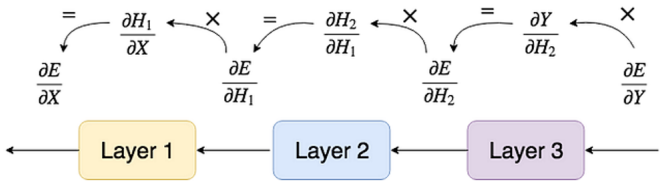



FIGURE 7 – Illustration de la rétro-propagation avant dans un réseau de neurones.

La méthode doit suivre ces étapes :

1. Parcourir les couches du réseau, en partant de la dernière couche et en remontant jusqu'à la première couche.
2. L'erreur doit être mise à jour après chaque appel de la méthode `backward` pour chaque couche. L'erreur doit être transmise à la couche précédente. À la fin de la méthode, renvoyez l'erreur finale.

## Espace de réponse pour Q16

```
def backward(self,e):
```

**Q17**. `compute_loss(...)` : Calcule et retourne `e` la perte entre les prédictions du modèle (`y_pred`) et les valeurs réelles (`y_true`) lors de l'entraînement du modèle. La perte est quantifiée selon la métrique **Categorical Crossentropy**. Définie par l'équation suivante :

$$E(y_{\text{pred}}, y_{\text{true}}) = -\frac{1}{\text{batch\_size}} \sum_{i=1}^{\text{batch\_size}} \sum_{j=1}^{\text{vocab\_size}} \left[ y_{\text{true}_{(i,j)}} \cdot \log\left( y_{\text{pred}_{(i,j)}} \right) \right] \tag{13}$$

Avec :

$y_{\mathbf{true}_{(i,j)}}$ **:** valeur réelle de la classe pour l'exemple $i$.

$y_{\mathbf{pred}_{(i,j)}}$ **:** probabilité prédite par le modèle pour la classe $j$ de l'exemple $i$.

**`vocab_size` :** nombre total de catégories (`output_size` de la dernière couche du réseau = nombre de colonnes de `y_pred`).

**`batch_size` :** nombre de lignes de `y_pred`.

### Espace de réponse pour Q17

```
def compute_loss(self,y_true, y_pred):
```

**Q18**. `compute_accuracy(...)` : Permet de calculer et retourner le pourcentage de prédictions correctes par rapport au nombre total d'exemples. L'accuracy est calculée comme suit :

- `y_pred` est transformée en une matrice binaire (0 ou 1) où 1 correspond à la probabilité maximale par ligne et 0 aux autres.

### Illustration sur exemple

$$\texttt{y\_pred} = \begin{bmatrix} \boxed{0.5} & 0.2 & 0.2 & 0.1 \\ 0.1 & \boxed{0.8} & 0.1 & 0.0 \\ 0.2 & 0.2 & 0.1 & \boxed{0.5} \end{bmatrix} \Longrightarrow \texttt{y\_pred} = \begin{bmatrix} \boxed{1} & 0 & 0 & 0 \\ 0 & \boxed{1} & 0 & 0 \\ 0 & 0 & 0 & \boxed{1} \end{bmatrix}$$

- La matrice binarisée est ensuite utilisée pour calculer l'accuracy en appliquant l'équation suivante :

$$\text{Accuracy} = \frac{\sum_{i=1}^{\text{batch\_size}} \delta(y_{\text{true},i} = y_{\text{pred},i})}{\text{batch\_size}} \tag{14}$$

où :

**`batch_size`** : nombre total d'exemples (nombre de lignes de $y_{\text{pred}}$).

$\delta(c)$ : est la fonction indicatrice, qui vaut 1 si la condition $c$ est vraie et 0 sinon.

$y_{\textbf{true},i}$ : est la classe correcte de l'exemple $i$.

$y_{\textbf{pred},i}$ : est la classe prédite pour l'exemple $i$ (après binarisation comme expliqué ci-haut).

### Espace de réponse pour Q18

```
def compute_accuracy(self, y_true, y_pred):
```

**Q19**. `fit(...)` : La méthode fit d'un modèle d'apprentissage est responsable d'entraîner le réseau en ajustant les poids et les biais de manière itérative. Elle utilise les données d'entraînement pour propager les informations avant, calculer la perte et son gradient relativement à la sortie, effectuer la rétro-propagation qui met à jour les paramètres selon la descente de gradient.

**Paramètres :**

- `x` : Les données d'entrée pour l'entraînement, sous forme d'un vecteur d'indices (de taille `n_samples`).
- `y_true` : les indices des tokens contexte pour chaque token cible de `x`. organisée dans une matrice de shape `(n_samples, vocab_size)`.
- `batch_size` : La taille des batchs utilisées lors de l'entraînement.
- `epochs` : Nombre d'époques pour entraîner le modèle (Le nombre total de passes sur l'ensemble des données d'entraînement.)

**Démarche :**

1. Les données d'entraînement représentées par un vecteur (`x`) et la matrice (`y_true`) doivent être divisées en batchs de taille `batch_size` (N.B la matrice `y_true` est découpée relativement aux lignes uniquement). Cette étape doit produire une liste `x_batches` de vecteurs et une liste `y_batches` de matrices.

### Illustration sur exemple

Pour

```
>>> vocab_size = 5
>>> x =  np.array([3, 4, 4, 3, 2, 4, 1, 3, 1, 4])
>>> y_true = np.array([[0., 0., 1., 0., 0.],
                       [0., 0., 0., 0., 1.],
                       [0., 0., 1., 0., 0.],
                       [0., 0., 0., 0., 1.],
                       [0., 0., 1., 0., 0.],
                       [0., 0., 0., 0., 1.],
                       [0., 0., 0., 0., 1.],
                       [0., 0., 0., 1., 0.],
                       [0., 1., 0., 0., 0.],
                       [0., 0., 0., 1., 0.]])

>>> batch_size = 3
```

On obtient

```
>>>  x_batches
[array([3, 4, 4]), array([3, 2, 4]), array([1, 3, 1]), array([4])]

>>> y_batches
[
    array([[0., 0., 1., 0., 0.],
           [0., 0., 0., 0., 1.],
           [0., 0., 1., 0., 0.]]),

    array([[0., 0., 0., 0., 1.],
           [0., 0., 1., 0., 0.],
           [0., 0., 0., 0., 1.]]),

    array([[0., 0., 0., 0., 1.],
           [0., 0., 0., 1., 0.],
           [0., 1., 0., 0., 0.]]),

    array([[0., 0., 0., 1., 0.]])
]
```

2. Le modèle passe plusieurs fois sur l'ensemble de données d'entraînement. Le nombre de passage est spécifié dans le paramètre `epochs`.
   (a) Chaque `x_batches[i]` est passé à travers les couches du réseau pour générer les prédictions (`y_pred`) en utilisant la méthode `forward`.
   (b) La fonction de perte est appliquée entre les prédictions (`y_pred`) et les étiquettes réelles (`y_batches[i]`). Cela permet de mesurer l'écart entre les résultats prévus et les résultats attendus. Le résultat obtenu est stocké dans une liste `epoch_loss`.
   (c) L'accuracy est calculée pour `x_batches[i]` et stockée dans une liste `epoch_accuracy`.
   (d) Le gradient de la perte est calculé selon l'équation :

$$\texttt{error} = \texttt{y\_pred} - \texttt{y\_batch[i]} \tag{15}$$

   (e) La rétro-propagation est effectuée à l'aide de la méthode `backward` pour calculer les gradients des pertes par rapport aux poids et biais du réseau.
3. À la fin de chaque époque, la perte moyenne et l'accuracy moyenne sont calculées et stockées respectivement dans les listes `loss_list` et `accuracy_list`).
4. La méthode retourne les listes `loss_list` et `accuracy_list`

Compléter les trous introduits dans le script de la méthode `fit(...)` selon les commentaires juxtaposés :

### Espace de réponse pour Q19

```
def fit(self, x,y_true,epochs,batch_size):

    n_samples = ..... #à compléter

    loss_list=[]
    accuracy_list=[]

    #à compléter
    x_batches = [..... for i in range(0, n_samples, batch_size)]

    #à compléter
    y_batches = [..... for i in range(0, n_samples, batch_size)]

    for epoch in range(epochs):
        epoch_loss=[]
        epoch_accuracy=[]
        for i in range(len(x_batches)):

            x_batch = ..... #à compléter

            y_batch = ..... #à compléter

            y_pred = ..... #à compléter

            l = self.compute_loss(y_batch, y_pred)
            epoch_loss.append(l)
            a = self.compute_accuracy(y_batch, y_pred)
            epoch_accuracy.append(a)
            error = y_pred - y_batch

            self.backward(.....) #à compléter

        loss_list.append(.....) #à compléter

        accuracy_list.append(.....) #à compléter

    return loss_list,accuracy_list
```

## Étape IV : Simulation du modèle Skip-Gram

Cette partie consiste à orchestrer les classes précédentes pour construire l'architecture du modèle Skip-Gram l'entrainer puis le tester sur un corpus.

**Q20**. Écrire les instructions Python permettant de réaliser les tâches suivantes :

1. Ouvrir un fichier texte nommé `"MonCorpus.txt"` en mode lecture, et stocker dans la variable `corpus` la liste des lignes contenues dans ce fichier.

2. Créer une instance de la classe `Tokenizer` et l'affecter à un objet nommé `tk`.
3. Initialiser les variables `window_size`, `embedding_dim`, et `learning_rate` avec les valeurs respectives `2`, `50`, et `0.01`.
4. Préparer l'objet `Tokenizer` `tk` à partir de la liste `corpus` en appelant la méthode `fit_on_texts`.
5. Calculer la taille du vocabulaire et l'affecter à la variable `vocab_size`.
6. Extraire les indices des `token_cibles` et la matrice (encodée en one-hot) des indices des `token_context`, puis les stocker respectivement dans les variables `x` et `y_true`.
7. Créer une instance de la classe `NeuralNetwork` nommée `skipgram` en utilisant le paramètre `learning_rate`.
   (a) Instancier la classe `Embedding` avec les paramètres `vocab_size` et `embedding_dim`, puis affecter l'instance à la variable `e_layer`.
   (b) Instancier la classe `Dense` avec les paramètres `embedding_dim` et `vocab_size`, puis affecter l'instance à la variable `d_layer`.
   (c) Ajouter les deux couches `e_layer` et `d_layer` au modèle `skipgram`.
8. Entraîner le modèle `skipgram` en utilisant les variables `x` et `y_true` avec les paramètres `epochs = 200` et `batch_size = 10`.
9. Tracer les courbes représentant la perte moyenne et l'accuracy moyenne pour chaque `epoch`.

## Espace de réponse pour Q20

```
import matplotlib.pyplot as plt
```

# Partie II : Base de Données Relationnelle

Le diagnostic du cancer, sous toutes ses formes, s'appuie sur un large éventail de techniques d'imagerie médicale avancées, conçues pour détecter avec précision d'éventuelles anomalies tissulaires. Parmi ces outils essentiels, on retrouve la mammographie, l'échographie, l'IRM, la tomodensitométrie (TDM) et la radiographie, chacune offrant des perspectives spécifiques selon la nature et la localisation de la lésion suspectée. Ces examens sont souvent complétés par une biopsie guidée par imagerie afin de confirmer la présence de cellules cancéreuses. Grâce aux progrès constants dans le domaine de l'imagerie médicale, les médecins disposent aujourd'hui de moyens performants pour affiner leurs diagnostics et adapter les traitements aux besoins spécifiques de chaque patient.

Le schéma de la base de données relationnelle qui suit fait partie du système d'information qui aide un médecin radiologue ou oncologue ou pathologiste à dépister un cancer, anticiper son traitement et épargner sa lourde prise en charge.

Le schéma relationnel de la base de données décrit ci-dessous a été élaboré pour les besoins de l'énoncé.

## Base de données Carcino-.

- `Patients`(`idP`, `NomP`, `DnaissP`, `SexeP`, `VilleP`, `MetierP`)
  La table `Patients` enregistre les informations relatives aux patients qui viennent consulter par des techniques d'imagerie.
  - `idP` : identifiant d'un patient de type entier clé primaire.
  - `NomP` : nom du patient, de type chaîne de caractères.
  - `DnaissP` : date de naissance du patient, de type date selon le format `"AAAA-MM-JJ"`.
  - `SexeP` : sexe du patient, de type caractère (`"F"` : Féminin, `"M"` : Masculin).
  - `VilleP` : ville du patient, de type chaîne de caractères.
  - `MetierP` : métier du patient, de type chaîne de caractères.
- `Medecins`(`idM` , `NomM`, `SpecM`, `EmailM`, `TelM`)
  La table `Medecins` enregistre les informations relatives aux médecins qui traitent les patients.
  - `idM` : identifiant d'un médecin, de type entier clé primaire.
  - `NomM` : nom du médecin, de type chaîne de caractères.
  - `SpecM` : spécialité du médecin, de type chaîne de caractères (ex. : `"Radiologue"`, `"Oncologue"`).
  - `EmailM` : adresse e-mail du médecin, de type chaîne de caractères.
  - `TelM` : numéro de téléphone du médecin, de type chaîne
- `Imageries`(`idI`, `TypeI`, `DateI`, #`idP`, #`idM`)
  La table `Imageries` enregistre les informations relatives aux examens d'imagerie réalisés pour chaque patient.
  - `idI` : identifiant d'une imagerie de type entier clé primaire.
  - `TypeI` : type d'imagerie réalisée (ex. : Mammographie, Échographie, IRM (Imagerie par Résonance Magnétique), TDM (Tomodensitométrie) et Radiographie, Biopsie à l'aiguille), de type chaîne de caractères.
  - `DateI` : date de l'examen d'imagerie, de type date selon le format `"AAAA-MM-JJ"`.

— `idP` : identifiant du patient concerné, de type entier clé étrangère fait référence à la table Patients.

— `idM` : identifiant du médecin ayant ordonné l'imagerie, de type entier clé étrangère fait référence à la table Medecins.

■ `Staff`(#`idM`, #`idI`)
La table `Staff` enregistre les associations entre les médecins qui ont collaboré avec le médecin donneur d'ordre de l'imagerie sur une imagerie spécifique pour se concerter et prendre une décision de traitement.

— `idM` : identifiant d'un médecin, de type entier clé étrangère fait référence à la table Medecins.

— `idI` : identifiant d'une imagerie, de type entier clé étrangère fait référence à la table Imageries.

La combinaison des clés étrangères `idM` identifiants du médecin et `idI` de l'imagerie forment la Clé primaire de la table, garantissant l'unicité de chaque collaboration.

■ `GroupementCellulaires`(`idGC`, `X`, `Y`, `Largeur`, `Longueur`, `Profondeur`, `Couleur`, `Texture`, `Douteux`, #`idI`)
La table `GroupementCellulaires` enregistre les informations sur les groupements cellulaires observés lors des examens d'imagerie.

— `idGC` : identifiant d'un groupement cellulaire, de type entier clé primaire.

— `X` : coordonnée X du groupement, de type réel.

— `Y` : coordonnée Y du groupement, de type réel.

— `Largeur` : largeur du groupement, de type réel.

— `Longueur` : longueur du groupement, de type réel.

— `Profondeur` : profondeur du groupement, de type réel.

— `Couleur` : couleur du groupement, de type chaîne de caractères.

— `Texture` : texture du groupement, de type chaîne de caractères.

— `Douteux` : indique si le groupement est suspect, de type booléen ( `True` ou `False`).

— `MasqueROI` : chemin d'accès à un fichier contenant un masque qui délimite le groupement cellulaire, de type chaîne de caractères.

— `idI` : identifiant de l'imagerie, de type entier clé étrangère fait référence à la table Imageries.

**NB :**

— `X`, `Y`, `Largeur`, `Longueur`, `Profondeur` sont exprimées en millimètre par rapport aux coordonnées de l'organe examiné.

— Le fichier contenu dans `MasqueROI` contient un masque couvrant la région d'intérêt marquant le groupement cellulaire. Se présente sous forme d'une matrice booléenne, $M_{yx} =$ `True` si les coordonnées $(y, x)$ appartiennent au groupement cellulaire.

■ `Diagnostics`(`idD`, `Decision`, `PCMD`, `RapportD`, `DateD`, `Recommandation`, #`idI`)
La table `Diagnostics` enregistre les informations relatives aux diagnostics effectués sur les imageries.

— `idD` : identifiant unique du diagnostic, de type entier clé primaire.

— `Decision` : diagnostic principal global (ex. : Bénin, Malin), de type chaîne de caractères.

— `PCMD` : pourcentage de cellules malines observées dans la totalité de l'imagerie, de type réel.

— `RapportD` : le rapport élaboré par le staff ou le médecin responsable détaillant le diagnostic, de type chaîne de caractères.
— `DateD` : date à laquelle l'analyse a été réalisée, de type date selon le format `"AAAA-MM-JJ"`.
— `Recommandation` : suggestions de chirurgie ou tout autre examen complémentaire pour le patient, de type chaîne de caractères.
— `idI` : identifiant de l'imagerie associée, de type entier, clé étrangère qui fait référence à la table Imageries.

# Travail demandé

## Algèbre relationnelle

Dans la suite on suppose que les tables de la base de données sont remplies en respectant les contraintes et les règles de gestion su citées.

Exprimer en algèbre relationnelle les requêtes permettant de :

**Q21**. Lister les métiers des patients de la ville de `"Zarzis"`.

### Espace de réponse pour Q21

**Q22**. Lister les identifiants des patients qui n'ont jamais effectué une imagerie de type `"IRM"`.

### Espace de réponse pour Q22

## SQL

**Q23**. La requête algébrique suivante permet de lister les noms des médecins ayant participé à des staffs pour des imageries dont les diagnostics ont abouti à des `PCMD` supérieur à 50%. On demande de traduire la requête algébrique donnée en SQL.

$$\prod_{\texttt{NomM}} \left( \sigma_{\texttt{PCMD}>50.0} \left( \texttt{Medecins} \bowtie_{\texttt{idM}} \texttt{Staff} \bowtie_{\texttt{idI}} \texttt{Diagnostics} \right) \right)$$

### Espace de réponse pour Q23

**Q24**. Pour chaque identifiant d'imagerie calculer selon l'ordre décroissant, le total de médecins qui ont fait staff pour l'examiner. Présenter le résultat par ordre décroissant du total des médecins.

### Espace de réponse pour Q24

**Q25**. Lister les villes où on recense la présence de plus de 30 cas de femmes pour lesquelles on a abouti à des décisions d'imageries autre que (`"Bénin"` et `"Normal"`).

### Espace de réponse pour Q25

**Q26**. Déterminez tous les détails des imageries qui pour le moment n'ont pas de diagnostic.

### Espace de réponse pour Q26

**Q27**. Lister les noms et les âges des patients pour qui les imageries ont nécessité des staffs de plus de trois médecins pour dépister leurs cas.
Il est conseillé de s'inspirer de la requête suivante qui utilise une fonction prédéfinie `julianday(date)` pour calculer les âges des patients à partir de leurs dates de naissance.

```sql
SELECT  (julianday('now') - julianday(date_naissance)) / 365 AS Age
FROM Patients;
```

### Espace de réponse pour Q27

**Q28**. Lister pour chaque identifiant d'imagerie le nombre de groupement cellulaires qui ont l'état douteux.

### Espace de réponse pour Q28

## SQLITE

L'objectif dans cette partie est de récupérer les données des diagnostics des patients depuis la base de données SQLite `"carcino-"` , de les structurer par ville et sexe, puis de générer un tableau de statistique présentant, pour chaque ville, le nombre de cas masculins et le nombre de cas féminins pour chaque type de diagnostic (`"Malin"`, `"Bénin"`, `"Normal"`), ainsi que le dénombrement des patients correspondants.

Dans la suite, les fonctions demandées doivent être écrites en Python en désignant par `cur` le curseur d'exécution des requêtes.

**Q29**. Écrire une fonction `villesPatients` qui prend en entrée `cur` le curseur de la connexion vers la base de données. La fonction retourne un ensemble contenant toutes les villes des patients.

### Espace de réponse pour Q29

```
def villesPatients(cur):
```

**Q30**. Écrire une fonction `decisionsDiagnostics` qui prend en entrée `cur` le curseur de la connexion vers la base de données. La fonction retourne un ensemble contenant toutes les décisions prises pour les diagnostics des patients.

### Espace de réponse pour Q30

```
def decisionsDiagnostics(cur):
```

**Q31**. Écrire une fonction `denombrerCas` qui prend en entrée :

- `cur` le curseur de la connexion vers la base de données.
- `ville` une chaine contenant la ville des patients.
- `sexe` un caractère contenant le sexe des patients ('F' ou 'M').
- `decision` une chaine contenant une décision de diagnostic.
- `annee` un entier contenant une année.

La fonction retourne le nombre de patients de la ville `ville` de sexe `sexe` qui ont été diagnostiqués avec la décision `decision` durant l'année `annee` (l'attribut `DateD` de la table `Diagnostics`) .

### Espace de réponse pour Q31

```
def denombrerCas(cur, ville, sexe, decision, annee):
```

**Q32**. Écrire une fonction nommée `statsAnnee` qui prend en entrée :

- `cur` le curseur de la connexion à la base de données.
- `annee` un entier contenant une année.

La fonction doit retourner un dictionnaire où :

— chaque clé est un tuple de la forme (ville, sexe, decision).

— chaque valeur est le nombre de cas qui ont été diagnostiqués dans cette ville, pour ce sexe et pour cette décision.

Le dictionnaire doit contenir toutes les combinaisons possibles de villes, de sexes et de décisions qui existent dans la base de données.

### Espace de réponse pour Q32

```
def statsAnnee(cur, annee):
```

## Espace de réponse supplémentaire

## Espace de réponse supplémentaire

# Annexe

## Quelques Fonctions/Méthodes Python/Schéma Relationnel

**Sans aucune obligation, les fonctions suivantes pourraient vous être utiles.**

### Opérations sur les itérables (`str`, `tuple`, `list`, `dict`, `etc.`)

- `len(it)` retourne le nombre d'éléments de l'itérable it.
- `range(d,f,p)` retourne la séquence des valeurs entières successives comprises entre $d$ et $f$, f exclu, par $pas = p$.
- `sum(it)` retourne la somme de tous les éléments de l'itérable it.
- `min(it)` retourne la valeur minimale de l'itérable it.
- `min(it, key = fct)` retourne la valeur minimale de l'itérable it où la fonction key définit le critère de comparaison.
- `max(it)` retourne la valeur maximale de l'itérable it.
- `max(it, key = fct)` retourne la valeur maximale de l'itérable it où la fonction key définit le critère de comparaison.
- `sum(it)` retourne la somme des éléments de l'itérable it.
- `x in it` vérifie si x appartient à it.
- `sorted(it)` retourne une liste contenant les éléments de it dans l'ordre croissant.
- `sorted(it, key = fct, reverse = True)` retourne une liste contenant les éléments de it dans l'ordre décroissant où la fonction key définit le critère de comparaison.
- `lst.sort()` trie la liste lst dans l'ordre croissant.
- `lst.sort(key = fct, reverse = True)` trie la liste lst en ordre décroissant en utilisant key comme critère de comparaison.
- `lst.count(val)` retourne le nombre d'occurrences de val dans la liste lst.
- `lst.append(val)` ajoute val à la fin de la liste lst.
- `lst.remove(val)` supprime la première occurrence val de la liste lst.
- `lst.pop(idx)` supprime puis retourne la valeur située à la position `idx` de la liste lst.
- `lst.index(val)` retourne l'indice de la première occurrence val de la liste lst.
- `chaine.lower()` retourne une chaîne de caractères où toutes les lettres majuscules sont converties en lettres minuscules.
- `chaine.upper()` retourne une chaîne de caractères où toutes les lettres minuscules sont converties en lettres majuscules.
- `chaine.isalphanum()` retourne `True` si la chaîne est composée uniquement de caractères alphanumériques (lettres et chiffres), et `False` sinon.
- `chaine.replace(ancien, nouveau)` retourne une chaîne de caractères où toutes les occurrences de `ancien` sont remplacées par `nouveau`.
- `source.split(motif)` retourne une liste formée par des chaînes de caractères résultantes du découpage de la chaîne source autour de la chaîne motif.

### Opérations sur les fichiers :

- `f=open(nomF,m)` permet d'ouvrir le fichier `nomF` en mode `m` ou `m='r'` ou `'w'`.
- `f.close()` permet de fermer un fichier.
- `f.read()` permet de lire et retourner le contenu d'un fichier dans une chaîne de caractères.

— `f.readline()` permet de lire et retourner la ligne courante d'un fichier dans une chaîne de caractères.
— `f.readlines()` permet de lire et retourner le contenu de toutes les lignes d'un fichier dans une liste.

**Module numpy**

— `M.shape` ou `np.shape(M)` retourne un `tuple` formé par le nombre de lignes et le nombre de colonnes d'une matrice M.
— `np.array(lst)` construit une instance de la classe `np.ndarray` dont les composantes sont initialisées à partir de la liste passée en entrée.
— `np.zeros(tpl)` construit une instance de la classe `np.ndarray` dont les composantes sont initialisées par zéro ayant un `shape` comme décrit par le `tuple` `tpl`.
— `np.dot(a, b)` effectue le produit matriciel/scalaire des tableaux `a` et `b`.
— `a.sum()` retourne la somme de tous les éléments dans l'objet `a` (généralement un tableau `numpy`).
— `a.sum(axis=axe)` retourne la somme des éléments le long de l'axe spécifié dans `axe`. Par exemple, `axis=0` somme les éléments par colonne, tandis que `axis=1` les somme par ligne.
— `np.exp(a)` retourne un tableau où chaque élément est la valeur exponentielle de l'élément correspondant dans `a`, c'est-à-dire `e**a[i]`.
— `a.copy()` retourne une copie indépendante de l'objet `a`. Toute modification apportée à la copie n'affectera pas l'original.
— `np.all(a)` retourne `True` si tous les éléments de l'objet `a` sont `True` (ou évalués comme vrais), et `False` sinon.
— `np.equal(a, b)` retourne un tableau booléen où chaque élément est `True` si les éléments correspondants de `a` et `b` sont égaux, et `False` sinon.

**Module `matplotlib.pyplot`**

— `plt.plot(x, y)` Tracer une courbe (ligne 2D) où x et y sont deux itérables contenant respectivement les abscisses et les ordonnées.

**Module sqlite3**

— `cur.execute(req)` exécuter la requête SQL décrite par le `str` req à partir du curseur `cur`.
— `cur.execute(req, seq)` exécuter la requête paramétrée décrite en SQL par le `str` req à partir du curseur `cur`.
— `cur.fetchall()` récupère tous les résultats de la dernière requête exécutée avec le curseur `cur` et les retourne sous forme de liste de tuples.

**Schéma relationnel BD**

- Patients(idP, NomP, DnaissP, SexeP, VilleP, MetierP)
- Medecins(idM , NomM, SpecM, EmailM, TelM)
- Imageries(idI, TypeI, DateI, #idP, #idM)
- Staff(#idM, #idI)
- GroupementCellulaires(idGC, X, Y, Largeur, Longueur, Profondeur, Couleur, Texture, Douteux, #idI)
- Diagnostics(idD, Decision, PCMD, RapportD, DateD, Recommandation, #idI)