TP N° 2 : Introduction à la programmation en langage Python

Préliminaire :

L'ensemble des fichiers que vous allez modifier est disponible sur github.

1. **Pour les récupérer, il vous faut installer Git (`https://git-scm.com/downloads`).**
2. **En fonction du système que vous utilisez, procédez à l'installation de la distribution.**
3. **Ensuite, dans le terminal : faites git clone https ://github.com/EDJINEDJA/TP2.git.**
4. **Pour engine004, j'estime que vous êtes déjà prêt pour découvrir le monde de l'ingénierie, organisez vos codes et résolvez le problème.**

There we go !

Exercice 1. Engine001-Module Numpy et Matplotlib (animation)

Dans cet exercice, nous allons utiliser le sous module **animation** de matplotlib

$$f(x,t) = e^{-(x-3t)^2}\sin(4\pi(x-t))$$

Soite f(x, t), une fonction qui modélise la fréquence cardiaque d'un patient au fil du temps t.

1- Écrivez un programme en Python qui visualise et anime f(x, t) dans l'intervalle $x \in [0, 10]$ pour $t \in [0, 3]$.

2- Enregistrez le rendu dans un fichier **.gif** à l'aide de l'instruction **animation.save**.

Exercice 2. Engine002-Numpy (self attention)

Le mécanisme de "self-attention" est utilisé par de grands modèles de langue pour générer du texte de manière cohérente.

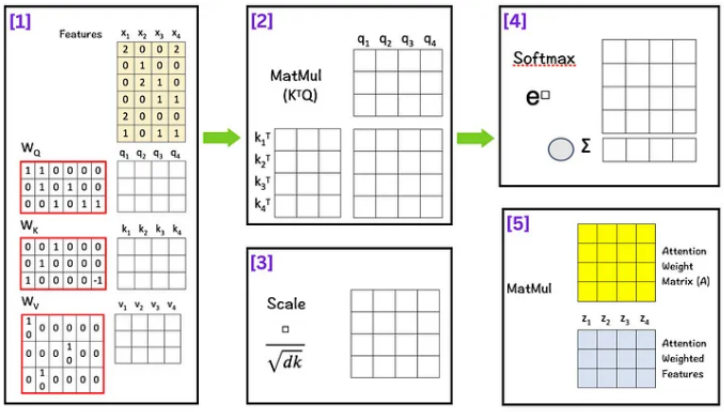

FIGURE 1 – Mécanisme de Self-Attention

Pour obtenir la matrice pondérée d'attention A, nous allons suivre les étapes de calcul décrites ci-dessous.

1. Intégration de mots

L'intégration de mots est une technique utilisée en traitement du langage naturel pour représenter les mots d'un texte sous forme de vecteurs numériques. Cela permet aux ordinateurs d'effectuer des opérations algébriques sur ces vecteurs afin d'en déduire des significations ou des relations.

Nous allons intégrer la phrase P1 : "je suis très malade".

1- Utilisez le module Numpy (randint) pour générer 4 vecteurs (X_1, X_2, X_3, X_4), chacun de dimension (1,6), correspondant aux mots respectifs "je", "suis", "très" et "malade".

2- Concaténez ces vecteurs en utilisant `np.vstack` pour former la matrice de caractéristiques F.

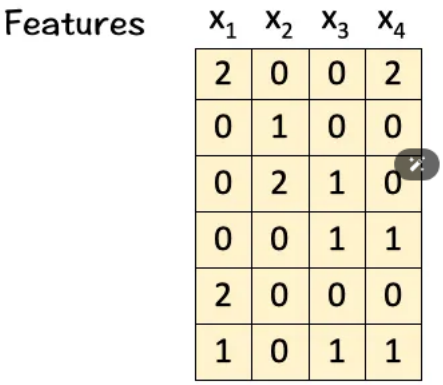

FIGURE 2 – Intégration de mots

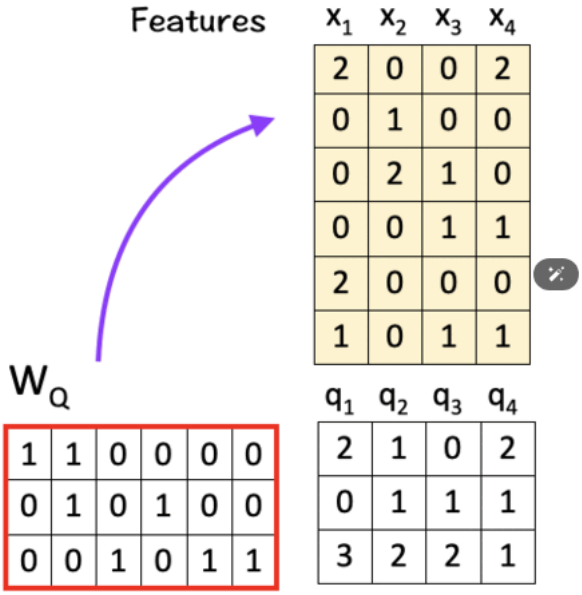

FIGURE 3 – Création de la matrice de requêtes

2. Création de la matrice de requêtes (Q)

Pour calculer la matrice de requêtes (q_1, q_2, q_3, q_4) associée à (X_1, X_2, X_3, X_4), effectuez un produit scalaire entre une matrice W_Q générée aléatoirement (avec `numpy.random.rand`) de dimension $(3, 6)$ et la matrice de caractéristiques F de dimension $(6, 4)$.

3. Création de la matrice de clés (K)

Pour calculer la matrice de clés (k_1, k_2, k_3, k_4) associée à (X_1, X_2, X_3, X_4), effectuez un produit scalaire entre une matrice W_K générée aléatoirement (avec `numpy.random.rand`) de dimension $(3, 6)$ et la matrice de caractéristiques F.

4. Création de la matrice de valeurs (V)

Pour calculer la matrice de valeurs (v_1, v_2, v_3, v_4) associée à (X_1, X_2, X_3, X_4), effectuez un produit scalaire entre une matrice W_V générée aléatoirement (avec `numpy.random.rand`) de dimension $(4, 6)$ et la matrice de caractéristiques F.

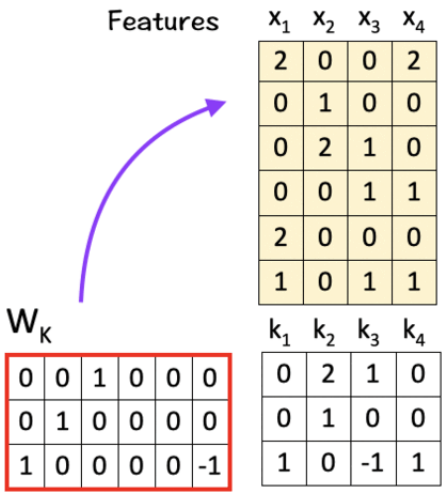

FIGURE 4 – Création de la matrice de clés

5. Calcul de la similarité cosinus entre chaque paire clé-requête

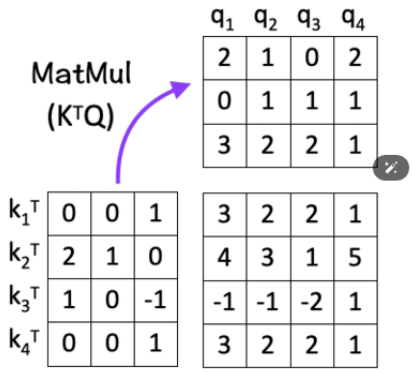

FIGURE 5 – Calcul de la similarité cosinus

Pour calculer la matrice de similarité cosinus pour chaque paire clé-requête, effectuez le produit scalaire entre la matrice K et Q.

6. Normalisation

Lors du calcul du produit scalaire entre chaque paire de vecteurs k et q, il peut y avoir une fuite de la variance selon la dimension de k. Pour éviter cela, normalisez la matrice résultante du produit scalaire entre K et Q en divisant chaque valeur de la matrice par $\sqrt{d_k}$, où $d_k = 3$ (faire l'arrondissement de $\sqrt{d_k}$).

7. Définition de la fonction Softmax

La fonction softmax transforme un vecteur de K nombres réels en une distribution de probabilités, permettant ainsi de choisir parmi K options.

La fonction softmax est définie comme suit :

$$\sigma : \mathbb{R}^K \to (0,1)^K, \quad \text{où } K \geq 1.$$

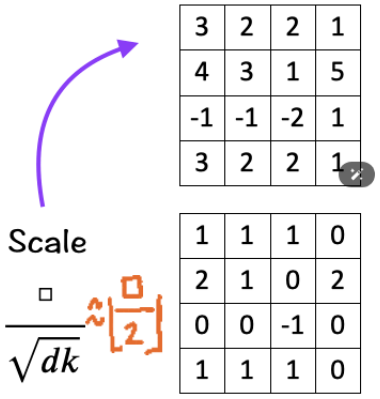

FIGURE 6 – Normalisation

Pour un vecteur $\mathbf{z} = (z_1, \ldots, z_K)$ de K nombres réels, la fonction softmax est calculée comme suit :

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}.$$

Pour chaque élément z_i du vecteur $\mathbf{z}$, cette formule permet de déterminer une composante correspondante dans le vecteur $\sigma(\mathbf{z})$, qui se situe entre 0 et 1.

Définissez une fonction `softmax` qui prend en entrée un vecteur de K nombres et renvoie une distribution de probabilités sur ces K choix.

8. Calcul de la matrice pondérée d'attention (A)

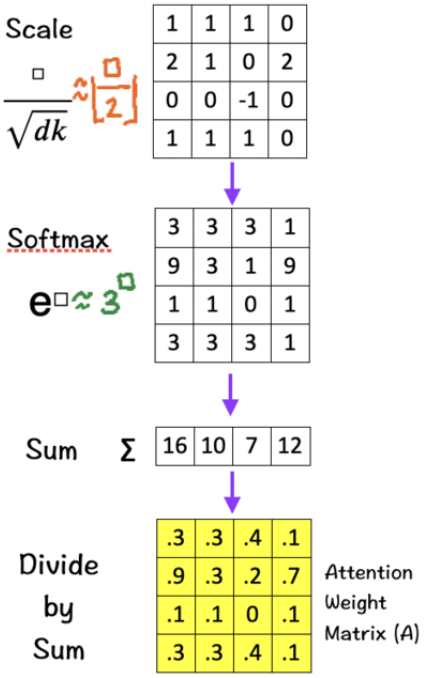

FIGURE 7 – Matrice pondérée d'attention

Pour chaque vecteur $K^T \cdot q_j$, appliquez votre fonction softmax. Utilisez la fonction `np.vstack` pour construire la matrice A.

Exercice 3. Engine003-Propagation d'un virus dans une population (Modèle SIR)

Le modèle SIR est utilisé pour modéliser la propagation d'une maladie dans une population. Il divise la population en trois compartiments :

— **S** (Susceptibles) : personnes qui peuvent contracter la maladie.
— **I** (Infectées) : personnes qui ont contracté la maladie et peuvent la transmettre.
— **R** (Rétablies) : personnes qui se sont rétablies et sont immunisées.

Les variations des populations S, I et R au fil du temps sont modélisées à l'aide du système d'équation suivante :

$$\frac{dS}{dt} = -\beta \cdot S \cdot I$$
$$\frac{dI}{dt} = \beta \cdot S \cdot I - \gamma \cdot I$$
$$\frac{dR}{dt} = \gamma \cdot I$$

Les paramètres que vous utiliserez dans la suite de l'exercice sont les suivants :
— β : Taux de transmission de la maladie (e.x., 0.3).
— γ : Taux de guérison (e.x., 0.1).
— S_0, I_0, R_0 : Proportions initiales de la population susceptible, infectée, et rétablie respectivement (e.x., $S_0 = 0.99$, $I_0 = 0.01$, $R_0 = 0.0$).
— max_time : Durée de la simulation (e.x., 109).

1- Créer une fonction récursive qui met à jour les valeurs de S, I et R à chaque appel en utilisant les équations ci-dessus. La fonction doit continuer jusqu'à ce que le temps maximal soit atteint ou que la population infectée devienne nulle.

2- Utiliser Matplotlib pour tracer les évolutions des populations S, I, et R au fil du temps. Assurez-vous de bien étiqueter vos axes et d'ajouter une légende.

Exercice 4. Engine004-Programmation Orienté Objet (POO)

À l'occasion d'un projet visant à automatiser un centre hospitalier pour aider le personnel de santé à faire face à des situations de crise sanitaire, vous êtes recruté en tant qu'ingénieur IA pour automatiser les processus d'activités à l'aide de robots.

Vous êtes chargé de créer les classes de robots et de les doter de différentes fonctions et caractéristiques afin qu'ils soient opérationnels.

1. Robots collaboratifs

Tous les robots sont des robots collaboratifs **RobotC**, définis par les caractéristiques suivantes :

1. Le numéro de série (Id) est un attribut privé de type entier.
2. La catégorie est un attribut public de type chaîne.
3. L'orientation est un attribut public de type entier, qui désigne l'orientation du robot (1 : NORD, 2 : EST, 3 : SUD, 4 : OUEST, 5 : NORD-EST, 6 : NORD-OUEST, 7 : SUD-EST, 8 : SUD-OUEST).
4. Le status est un attribut public de type booléen, qui renseigne sur l'état de fonctionnement du robot (True : En service, False : Hors service).
5. La position est un attribut de type **Point**, qui renseigne sur les coordonnées en 3D du robot (x : abscisse, y : ordonnée, z : coordonnée suivant l'axe z perpendiculaire aux axes x et y).

L'objet de type **Point** est lui-même une classe qui permet de définir les coordonnées initiales du robot dans l'espace. Ses attributs sont les suivants : x (abscisse), y (ordonnée) et z (coordonnée suivant l'axe z), et possède les méthodes suivantes responsables du déplacement d'un robot :

1. set_xyz(self, x, y, z) : pour initialiser les coordonnées du robot dans un espace en 3D.

2. deplace(self, dx, dy, dz, θ) : permet au robot de se déplacer dans un espace en 3D en effectuant une rotation d'angle θ autour de lui-même (axe Z), suivie d'une translation (dx, dy, dz).

Les méthodes que possèdent RobotC diffèrent d'un robot à un autre en fonction de leurs fonctionnalités, mais tous les robots collaboratifs **RobotC** possèdent les méthodes suivantes :

1. Constructeur qui possède la signature suivante : Constructor() -> RobotC ou Constructor(Id : int, Catégorie : str) -> RobotC.
2. Surcharge de l'opérateur ___str___ qui permet d'afficher les informations du robot et l'état de son fonctionnement.
3. getCatégorie() -> str : retourne la catégorie de robot.
4. getId() -> int : retourne le numéro de série du robot.
5. getPosition() -> Point : retourne la position du robot dans l'environnement.
6. getStatus() -> bool : retourne le status du robot.
7. setEtat(status : bool) : définit l'état du robot.
8. getOrientation() -> int : retourne l'orientation du robot.
9. setOrientation(orientation : int) : définit l'orientation du robot.
10. tourner(nouvelle_orientation : int) : permet de tourner le robot en fonction de l'orientation actuelle et de la nouvelle orientation.

Instanciez cette classe avec un tableau de 4 robots en utilisant l'ensemble des méthodes pour au moins l'un des quatre.

2. Robots collaboratifs mobiles

Parmi les robots collaboratifs, il existe les robots collaboratifs mobiles **(RobotCMobile)** qui possèdent des fonctionnalités supplémentaires en plus des fonctionnalités d'un robot collaboratif. Ils se caractérisent par les méthodes suivantes :

Une méthode avancer() qui permet d'avancer le robot selon son orientation :

1. Si on avance de dx vers l'Est, l'abscisse augmente de dx.
2. Si on avance de dx vers l'Ouest, l'abscisse diminue de dx.
3. Si on avance de dy vers le Nord, l'ordonnée augmente de dy.
4. Si on avance de dy vers le Sud, l'ordonnée diminue de dy.

3. Robots de surveillance

Les robots de surveillance sont des robots collaboratifs qui échangent des informations entre eux. En fonction de l'orientation des autres robots de surveillance, ils choisissent une autre orientation. La méthode supplémentaire qu'ils possèdent est :

1. sendOrientation() : permet d'envoyer son orientation aux autres robots de surveillance.

4. Robots d'accueil

Les robots d'accueil permettent d'accueillir les visiteurs, de leur fournir des informations et de les guider à travers l'hôpital. Ils disposent de méthodes telles que :

1. accueillir(self, visiteur : str) -> None : accueille un visiteur.
2. indiqueChemin(self, destination : str) -> None : indique le chemin vers une destination.
3. prendreRendezVous(self, details : str) -> None : prend un rendez-vous.

5. Robots d'assistance chirurgicale

Les robots d'assistance chirurgicale aident les équipes chirurgicales en préparant les instruments et en suivant les opérations. Leurs méthodes incluent :

1. preparerInstrument(self, instrument : str) -> None : prépare un instrument.
2. suivreOperation(self) -> None : suit une opération.
3. assisterChirurgien(self) -> None : assiste le chirurgien.

6. Robots de nettoyage

Les robots de nettoyage sont responsables de la propreté et de la désinfection des espaces hospitaliers. Ils possèdent des méthodes comme :

1. nettoyerZone(self, zone : str) -> None : nettoie une zone.
2. decontaminer(self) -> None : désinfecte la zone.
3. rapporterProbleme(self, probleme : str) -> None : signale un problème.

Références : Pour la simplification de l'exercice 2, certaines images sont prises du poste de (Srijanie Dey), disponible sur le site medium.