Université Paris-Saclay
UVSQ - Université de Versailles Saint-Quentin-en-Yvelines

Unité d'Enseignement : Apprentissage et Réseaux de Neurones

TP 5.1 : Réseaux de Neurones Récurrents et Analyse de Séquences

Génération de poésie par apprentissage de modèles de langage sur l'œuvre de Charles Baudelaire

Étudiant :
DEBACHE Mounsef

Enseignants :
M. Arnaud Breloy
M. Olivier Pons

Février 2026

Exercice 1(a) – Génération des données et étiquettes

Parsing et pré-traitement du texte. Le fichier `fleurs_mal.txt` est parcouru ligne par ligne. Chaque ligne est nettoyée par `strip()` (suppression des blancs en début/fin) puis convertie en minuscules (`lower()`). On ne conserve ensuite que les lignes non vides situées entre deux marqueurs :

- début du corpus lorsque la phrase `"charles baudelaire avait un ami"` est rencontrée (`START`),
- fin du corpus au marqueur `"End of the Project Gutenberg EBook of Les Fleurs du Mal, by Charles Baudelaire"` (`END`).

Les lignes retenues sont concaténées avec un espace pour former une chaîne unique :

$$\texttt{text} = \texttt{" ".join(lines2)}.$$

Dans nos résultats, on obtient :

$$|\texttt{text}| = 146\,176 \text{ caractères.}$$

Interprétation de `chars`. La variable `chars` est définie par :

$$\texttt{chars} = \text{sorted}\Big(\text{set}\big(\{c \mid c \in \texttt{text}\}\big)\Big).$$

Elle représente donc **l'ensemble des caractères distincts** présents dans le corpus (l'*alphabet* du texte), trié dans un ordre déterministe. Autrement dit, c'est le vocabulaire au niveau *caractère* utilisé pour la modélisation séquentielle.

Dans notre cas, `chars` contient notamment :

- l'espace `' '`,
- de la ponctuation (`! , . : ; ? ' ( ) - _` et les guillemets `« »`),
- des chiffres (`0,1,2,5,6,7,8`),
- les lettres latines minuscules `a` à `z`,
- des lettres accentuées (`à, â, ç, è, é, ê, ë, î, ï, ô, ù, û, ü`).

Interprétation de `nb_chars`. La variable `nb_chars` correspond au nombre de caractères uniques :

$$\texttt{nb\_chars} = |\texttt{chars}| = 60.$$

Elle donne donc la **taille de l'alphabet** du corpus. Dans un modèle de génération de texte au niveau caractère, `nb_chars` est directement la dimension :

- de la représentation *one-hot* des caractères,
- et de la couche de sortie (softmax) qui prédit le prochain caractère.

Exercice 1(b) : Apprentissage auto-supervisé d'un RNN pour la génération de texte

Rappel du problème (auto-supervision). On cherche à entraîner un réseau de neurones récurrent à **prédire le caractère suivant** d'un texte, à partir d'une fenêtre glissante de longueur fixée $SEQLEN$. Chaque exemple d'apprentissage est donc un couple :

$$x^{(i)} = (c_i, c_{i+1}, \ldots, c_{i+SEQLEN-1}), \qquad y^{(i)} = c_{i+SEQLEN},$$

où c_t est un caractère du texte. Il s'agit d'un apprentissage **auto-supervisé** : il n'existe pas de labels externes (comme en classification d'images), mais on construit la supervision en demandant au modèle de prédire le prochain caractère.

Données et labels. On utilise :

$$SEQLEN = 10, \qquad nb\_chars = 60$$

(où nb_chars est la taille du dictionnaire de caractères).

Après vectorisation one-hot, on obtient :

$$X \in \{0,1\}^{nbex \times SEQLEN \times nb\_chars}, \qquad y \in \{0,1\}^{nbex \times nb\_chars}.$$

Les dimensions observées sont :

$$X_{train} : (116933,\ 10,\ 60), \quad y_{train} : (116933,\ 60),$$

$$X_{test} : (29233,\ 10,\ 60), \quad y_{test} : (29233,\ 60).$$

Les données sont sauvegardées dans `Baudelaire_len_10.p`.

Architecture du modèle RNN. Le modèle Keras utilisé est :

- une couche récurrente `SimpleRNN` de taille cachée $HSIZE = 128$,
- une couche entièrement connectée `Dense(60)` produisant les scores pour chacun des 60 caractères,
- une activation `softmax` pour produire une distribution de probabilité.

On a donc :

$$h_t = \phi(W_{xh}x_t + W_{hh}h_{t-1} + b_h), \qquad \hat{y} = \text{softmax}(W_{hy}h_{SEQLEN} + b_y).$$

Sens de `return_sequences=False`. Dans Keras, `return_sequences=False` signifie que la couche `SimpleRNN` **retourne uniquement le dernier état caché** h_{SEQLEN} (un vecteur de dimension 128), et non pas la séquence complète $(h_1, \ldots, h_{SEQLEN})$. Ceci est cohérent avec notre tâche : **on ne prédit qu'un seul caractère** (le suivant) à partir de la fenêtre entière, donc on exploite un seul vecteur résumé de la séquence.

Nombre de paramètres (vérification avec `summary()`). Le résumé Keras indique :

- `SimpleRNN(128)` : 24 192 paramètres
- `Dense(60)` : 7 740 paramètres
- `Softmax` : 0 paramètre

Total :

$$\text{Total params} = 31\,932.$$

Apprentissage : fonction de coût et optimisation. Le modèle est entraîné avec :

- perte : `categorical_crossentropy` (classification multi-classes sur 60 caractères),
- optimiseur : `RMSprop` avec $\eta = 0.001$,
- `batch_size` = 128,
- `epochs` = 50,
- métrique : `accuracy`.

Courbes d'apprentissage. Les courbes montrent une amélioration progressive :

- l'accuracy d'entraînement augmente de ≈ 0.30 au début jusqu'à ≈ 0.53 à la fin,
- l'accuracy de validation / test augmente jusqu'à ≈ 0.46,
- la loss diminue, puis tend à plafonner (signe d'un apprentissage limité par la capacité du modèle ou l'ambiguïté intrinsèque du texte).

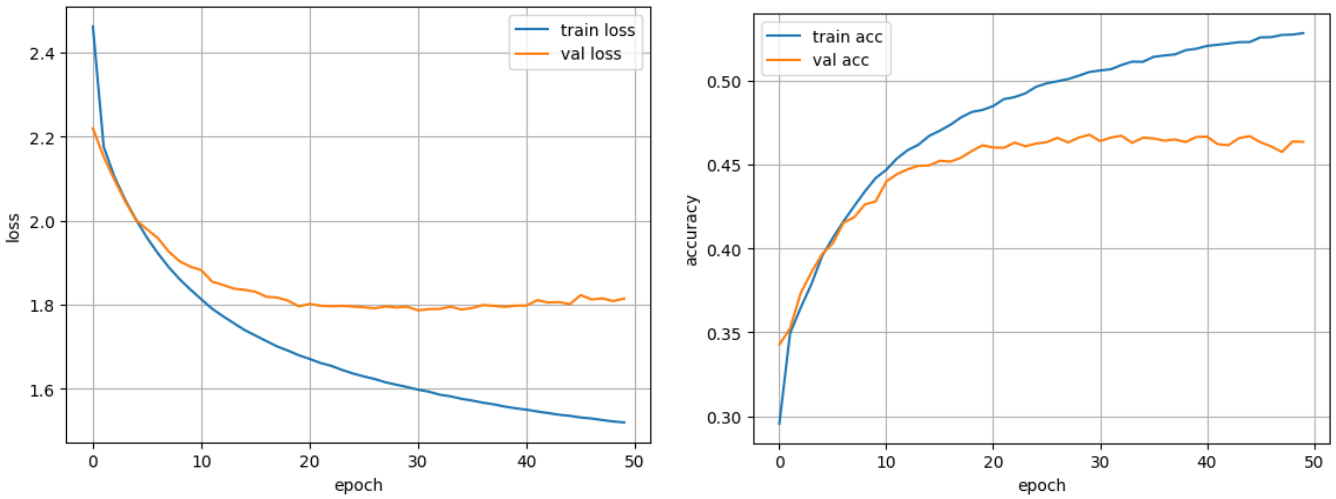

FIGURE 1 – Courbes d'apprentissage du modèle RNN : (gauche) loss ; (droite) accuracy (train/val) sur 50 époques.

Performances finales. Après entraînement, on obtient :

$$\text{Accuracy train} = 53.99\%, \qquad \text{Accuracy test} = 46.36\%.$$

Analyse des performances. Un taux de $\approx 46\%$ en test peut sembler faible comparé à MNIST, mais il s'explique par la nature de la tâche :

- La prédiction du caractère suivant est **ambiguë** : une même séquence de 10 caractères peut être suivie de plusieurs caractères possibles selon le contexte plus long (mot complet, phrase, ponctuation, etc.).
- Le modèle `SimpleRNN` avec $SEQLEN = 10$ a une mémoire limitée : il ne voit qu'un contexte très court, et ne capture pas des dépendances longues (syntaxiques / sémantiques).
- La distribution des caractères est **déséquilibrée** : l'espace " " (espace) ou certains caractères fréquents dominent, ce qui influence l'accuracy.

En quoi ce problème diffère d'une classification classique ? Dans une classification supervisée classique (ex : MNIST), chaque entrée x est associée à une **étiquette unique** y stable (ex : chiffre 0–9). Ici :

- les labels sont construits automatiquement à partir du texte,
- la fonction cible n'est pas déterministe au sens strict (plusieurs sorties plausibles),
- la tâche dépend fortement du **contexte séquentiel**.

Exemple : séquence `"la mort de "`. Avec $SEQLEN = 10$, on a recherché la séquence exacte :

`"la mort de "`

Dans le corpus, on observe :

$$\text{Nb occurrences} = 2, \qquad \text{labels cibles observés} = [\texttt{"a"}, \texttt{"e"}],$$

donc le caractère suivant est systématiquement un espace :

$$P(y = \texttt{" "} \mid x = \texttt{"la mort de "}) = 1 \quad \text{sur ces 2 occurrences.}$$

Interprétation. Cet exemple illustre que certaines séquences peuvent être très déterministes (toujours suivies d'un espace, d'une lettre particulière, etc.), mais en général :

— beaucoup de séquences apparaissent avec **plusieurs continuations possibles** (variations de mots, ponctuation, accords),

— plus $SEQLEN$ est petit, plus l'ambiguïté est forte.

Conclusion de la partie (b). Le RNN apprend à modéliser une partie des régularités locales du texte (orthographe, enchaînements fréquents, ponctuation), ce qui se traduit par une accuracy de l'ordre de 54% en apprentissage et 46% en test. Cependant, la génération de texte requiert souvent des dépendances plus longues que $SEQLEN = 10$; un modèle plus performant nécessiterait typiquement un contexte plus long et/ou des architectures plus adaptées aux longues dépendances (LSTM/GRU, ou modèles attentionnels).

(c) Génération de texte avec le modèle appris

Principe de la génération

Après l'entraînement du réseau récurrent, on souhaite générer du texte en itérant une prédiction caractère par caractère. Le modèle prend en entrée une séquence de longueur $SEQLEN = 10$ (encodée en one-hot sur un dictionnaire de $tdict = nb\_chars = 60$ symboles), et prédit une distribution de probabilité $p(\cdot \mid \text{séquence})$ sur le caractère suivant.

On fixe une séquence initiale (*seed*) extraite de X_{train} :

$$\text{char\_init} = (c_1, \ldots, c_{SEQLEN})$$

puis on génère *nbgen* nouveaux caractères en : (i) prédisant p, (ii) tirant un caractère selon p (échantillonnage), (iii) décalant la fenêtre d'entrée et en y ajoutant le caractère généré.

Température et renormalisation de la distribution (question)

Au lieu de prendre directement arg max (choix déterministe), on effectue un échantillonnage selon une version *tempérée* de la distribution. À partir des probabilités z_i (sortie softmax), on définit :

$$z_i^{(T)} = \frac{z_i^{1/T}}{\sum_{j=1}^{C} z_j^{1/T}}$$

où $T > 0$ est la température et $C = nb\_chars$.

Comportement limite :

— **Quand $T \to 0$:** la distribution devient **très piquée** (quasi déterministe). L'échantillonnage se rapproche d'un choix glouton (arg max). On obtient souvent des séquences très cohérentes localement, mais avec risque de répétitions et de manque de diversité.

— **Quand $T \to +\infty$:** la distribution se **raplatit** vers une loi presque uniforme. Les tirages deviennent plus aléatoires : la diversité augmente, mais le texte se dégrade (caractères incohérents, bruit).

Observation expérimentale de l'effet de T

Les essais réalisés confirment exactement ce comportement :

— $T = 0.1$ **:** texte très répétitif, fort biais vers le caractère le plus probable (max prob ≈ 0.99996).
— $T = 0.3$ **à** 0.5 **:** meilleur compromis : phrases pseudo-françaises, structure plus variée mais encore lisible.
— $T = 0.8$ **à** 1.0 **:** diversité plus grande, apparition d'artefacts (mots inventés, fautes).
— $T = 1.3$ **:** génération instable, nombreux caractères incohérents (ponctuation et symboles inattendus).

Ainsi, dans ce TP, une température intermédiaire ($T \approx 0.5$) est la plus adaptée pour générer un texte qui reste lisible tout en conservant une certaine créativité.

Impact du nombre d'époques (10, 30, 50, 70) à $T = 0.5$

On a entraîné le même modèle SimpleRNN (HSIZE=128) pendant différentes durées (10, 30, 50, 70 époques). Les courbes d'apprentissage (accuracy/loss) montrent :

— **Amélioration rapide** au début (jusqu'à environ 30 époques),
— puis **plateau** : l'accuracy validation se stabilise,
— et l'écart train/val augmente légèrement : signe d'un **début de sur-apprentissage** (overfitting léger).

Résumé quantitatif (val_accuracy observée en fin d'entraînement).

Époques	**Train acc (fin)**	**Val acc (fin)**
10	≈ 0.44	≈ 0.42
30	≈ 0.50	≈ 0.45
50	≈ 0.52	≈ 0.44
70	≈ 0.53	≈ 0.44

On observe que :

— la performance **validation** progresse jusqu'à ~ 30 époques,
— puis **n'augmente plus** (voire baisse légèrement), alors que l'accuracy train continue de monter.

Cela indique que le modèle apprend de mieux en mieux le corpus d'entraînement, mais ne généralise pas davantage sur des séquences non vues.

Exemples de générations (à $T = 0.5$, seed identique)

On compare des textes générés à température fixe ($T = 0.5$) mais pour différents nombres d'époques :

Après 50 époques :

> souvent, pour l'antre plein du contemplit à la mer tout de bénale enchante en minitée il tes grant les fantessel avent de coeur et matin. ...

(extrait issu du fichier de génération epochs=50) :contentReference[oaicite :2]index=2

Après 70 époques :

> souvent, passe soin de posteux, et comme pessieux comme le faut de l'aireil de chare vouvent le fleur que le fin ...

(extrait issu du fichier de génération epochs=70) :contentReference[oaicite :3]index=3

Analyse qualitative :

- Quand le nombre d'époques augmente, le modèle produit des fragments plus *structurés* localement (enchaînements plus plausibles), mais la cohérence globale reste limitée (pas de syntaxe complète).
- Le vocabulaire généré ressemble davantage au corpus (mots comme *coeur, douleur, pleurs, fleur*), ce qui montre que le réseau apprend des régularités statistiques.
- Cependant, le modèle est un SimpleRNN avec fenêtre courte ($SEQLEN = 10$) : il capture surtout des dépendances locales (orthographe approximative, accords faibles, répétitions).

Forces et faiblesses du générateur (question)

Points forts :

- Apprentissage **auto-supervisé** : aucun label manuel, la supervision est le caractère suivant.
- Le modèle apprend des **régularités** du style (ponctuation, espaces, motifs fréquents, fragments de mots).
- Contrôle de créativité via la température : petit T = texte conservateur, grand T = texte exploratoire.

Points faibles :

- Dépendances longues mal capturées : SimpleRNN + $SEQLEN = 10$ limite fortement la mémoire.
- Texte souvent non grammatical : beaucoup de **mots inventés** et de **fautes**.
- Sur-apprentissage possible au-delà d'un certain nombre d'époques : la val_accuracy plafonne.

Conclusion (partie c). La génération dépend fortement de la température : un T faible favorise la cohérence mais cause de la répétition, alors qu'un T élevé augmente la diversité au prix d'une perte de structure. De plus, augmenter les époques améliore le modèle jusqu'à un plateau (environ 30 époques ici), après quoi la généralisation n'augmente plus : on observe donc un compromis entre apprentissage et sur-apprentissage.

Exercice 2 — Embedding vectoriel de texte (GloVe) : extraction sur Flickr8k

Objectif

L'objectif est d'associer à chaque mot apparaissant dans les légendes du corpus Flickr8k un vecteur d'embedding GloVe. Ces représentations vectorielles seront utilisées dans le TP suivant pour encoder les légendes dans une tâche de légendage d'images. Deux tokens spéciaux, `<start>` et `<end>`, sont également ajoutés afin de délimiter le début et la fin d'une séquence de mots.

Extraction du vocabulaire des légendes

Le fichier d'apprentissage contient N légendes (une ligne par légende), chacune associée à une image. Les légendes sont tokenisées par découpage sur les espaces, puis converties en minuscules. On obtient l'ensemble des mots distincts :

$$V = \{w_1, \ldots, w_{|V|}\},$$

avec :

$$|V| = 8918.$$

Ce vocabulaire contient donc 8918 mots uniques extraits des légendes.

Filtrage du modèle GloVe

Le modèle utilisé est `glove.6B.100d.txt`, qui fournit un vecteur de dimension $d = 100$ pour chaque mot présent dans le fichier. On parcourt le fichier GloVe et on conserve uniquement les mots appartenant au vocabulaire V. On ajoute également le token `unk` afin de représenter les mots des légendes absents du modèle GloVe (hors-vocabulaire).

Le filtrage conduit à :

$$|V_{\text{GloVe}}| = 7977 \quad \text{et} \quad d = 100.$$

Autrement dit, 7977 mots des légendes disposent d'un embedding GloVe (dimension 100). La couverture (proportion de mots uniques des légendes trouvés dans GloVe) vaut :

$$\text{couverture} = \frac{|V_{\text{GloVe}}|}{|V|} = \frac{7977}{8918} \approx 0.894 \text{ soit } 89.4\%.$$

Construction de la matrice des embeddings

On construit une matrice d'embeddings E contenant les vecteurs GloVe, à laquelle on ajoute deux lignes pour les tokens `<start>` et `<end>`, ainsi que deux colonnes supplémentaires pour les coder explicitement.

La matrice est définie par :

$$E \in \mathbb{R}^{(|V_{\text{GloVe}}|+2)\times(d+2)}.$$

Avec les valeurs mesurées :

$$|V_{\text{GloVe}}| + 2 = 7977 + 2 = 7979 \quad \text{et} \quad d + 2 = 100 + 2 = 102,$$

donc :

$$\text{shape}(E) = (7979, 102).$$

Contenu des colonnes.

- Colonnes $0 \ldots 99$: vecteur GloVe de dimension 100.
- Colonne 100 : indicateur (flag) du token `<start>`.
- Colonne 101 : indicateur (flag) du token `<end>`.

Contenu des lignes.

- Lignes $0 \ldots 7976$: mots du vocabulaire filtré V_{GloVe} (embeddings GloVe).
- Ligne 7977 : token `<start>`.
- Ligne 7978 : token `<end>`.

Codage des tokens spéciaux. Les tokens `<start>` et `<end>` n'ont pas de vecteur sémantique GloVe ; leurs 100 premières composantes sont nulles. Ils sont distingués par les deux dernières colonnes :

$$E[7977, 100:102] = [1, 0], \qquad E[7978, 100:102] = [0, 1].$$

Ces valeurs ont été vérifiées expérimentalement (flags corrects).

Résultats expérimentaux et discussion

Couverture du vocabulaire. La couverture du vocabulaire par GloVe est d'environ 89.4% (7977 mots couverts sur 8918 mots uniques). Cette valeur est satisfaisante : la grande majorité des mots des légendes disposent d'une représentation vectorielle sémantique. Les $\approx 10.6\%$ restants correspondent à des mots absents du modèle (noms propres, variantes orthographiques, mots rares, etc.) et seront encodés par le token `unk` lors de la conversion des légendes en séquences d'indices.

Synthèse des dimensions. Le tableau 1 résume les tailles obtenues.

Quantité	Valeur
Nombre de mots uniques dans les légendes $\lvert V\rvert$	8918
Nombre de mots couverts par GloVe $\lvert V_{\text{GloVe}}\rvert$	7977
Dimension des embeddings GloVe d	100
Nombre total de tokens (avec `<start>`, `<end>`)	$7977 + 2 = 7979$
Taille finale de la matrice E	$(7979,\ 102)$
Couverture $\frac{\lvert V_{\text{GloVe}}\rvert}{\lvert V\rvert}$	≈ 0.894 (89.4%)

TABLE 1 – Récapitulatif des statistiques et dimensions pour l'extraction des embeddings GloVe sur Flickr8k.

Sauvegarde

Enfin, la liste des mots `listwords` et la matrice E sont sauvegardées dans un fichier binaire :

`Caption_Embeddings.p`,

via `pickle`, afin d'être réutilisées directement au TP suivant.

Analyse des embeddings GloVe des légendes (partie b)

Chargement des embeddings

Le fichier `Caption_Embeddings.p` est chargé et fournit :

$$\texttt{embeddings.shape} = (7979,\ 102), \qquad |\texttt{listwords}| = 7979.$$

Les deux derniers tokens du vocabulaire sont bien les symboles de séquence :

$$\texttt{listwords[-2:]} = [\texttt{<start>}, \texttt{<end>}].$$

Normalisation L2 des vecteurs

On normalise chaque vecteur d'embedding (partie sémantique GloVe, dimensions $0 \dots 99$) afin d'obtenir une norme euclidienne unitaire. Pour un mot représenté par un vecteur $v \in \mathbb{R}^{100}$, la normalisation L2 est définie par :

$$\hat{v} = \frac{v}{\|v\|_2} \quad \text{si } \|v\|_2 > 0, \qquad \text{avec } \|v\|_2 = \sqrt{\sum_{k=1}^{100} v_k^2}.$$

Objectif de la normalisation. La normalisation L2 poursuit plusieurs objectifs :

- **Comparer par direction (cosinus) plutôt que par amplitude :** après normalisation, la similarité entre mots dépend principalement de l'angle entre leurs vecteurs (information sémantique), et non de la norme.
- **Réduire les biais liés à la norme :** sans normalisation, des vecteurs de grande norme peuvent dominer la distance euclidienne et influencer excessivement l'algorithme de clustering.
- **Améliorer la stabilité du KMeans :** KMeans minimisant des distances euclidiennes, la normalisation rend la partition plus cohérente et moins sensible aux variations d'échelle.

Vérification expérimentale. Après normalisation, les normes observées (sur la partie GloVe) sont :

$$\min \|\hat{v}\|_2 = 0.0, \qquad \max \|\hat{v}\|_2 \approx 1.0.$$

Les tokens spéciaux `<start>` et `<end>` ont une norme nulle sur la partie GloVe :

$$\|\hat{v}_{\texttt{<start>}}\|_2 = 0.0, \qquad \|\hat{v}_{\texttt{<end>}}\|_2 = 0.0,$$

ce qui est attendu car ils ne possèdent pas d'embedding sémantique GloVe (leurs 100 premières composantes sont nulles et ils sont codés par des indicateurs).

Clustering KMeans dans l'espace des embeddings

Un clustering est effectué avec l'algorithme KMeans en $K = 10$ groupes, en utilisant :

$$\texttt{init} = \text{“random”}, \qquad \texttt{max\_iter} = 1000.$$

On obtient :

$$\texttt{clustersID.shape} = (7979), \qquad \texttt{clusters.shape} = (10,\ 102).$$

Pour chaque cluster, on affiche le mot le plus proche du centre (représentant) ainsi que les 20 mots suivants les plus proches.

Cluster (centre)	**20 mots les plus proches**
0 (lazily)	poking, playfully, flinging, crazily, crawling, gazes, suggestively, nibbling, peeking, splashing, rubbing, zipping, giggling, spits, tugging, gazed, gazing, squinting, whacking, ducking
1 (onto)	gear, wheel, vehicle, car, off, inside, cars, hand, used, speed, into, down, attached, using, front, up, door, wheels, device, carrying
2 (white-colored)	four-wheeler, leather-clad, hairnet, different-colored, bellbottoms, breakdancer, codpiece, rollerskates, zip-line, headress, grey-blue, half-completed, snowsuit, long-handled, green-colored, onesie, blown-up, bodyboard, lavendar, showerhead
3 (kind)	love, life, show, well, like, shows, look, so, sort, even, you, something, what, same, little, friends, she, how, one, fun
4 (cream)	bread, cake, vegetables, baked, cans, soup, coffee, plastic, meat, fruit, butter, chocolate, chicken, candy, milk, bags, drinks, stuffed, soda, cheese
5 (well)	even, because, though, but, only, so, this, same, not, the, way, they, one, it, yet, that, take, rather, some, would
6 (embroidered)	trousers, sleeves, multicolored, leggings, pants, oversized, worn, blouse, jacket, scarf, shiny, beaded, sweater, dresses, patterned, tunic, tights, sequined, striped, dyed
7 (puppy)	skateboard, kitten, pony, buggy, mule, poodle, cat, dachshund, rottweiler, sled, squirrel, bunnies, monkey, bulldog, ox, alligator, hound, surfboard, pug, donkeys
8 (hillside)	overlooking, beneath, courtyard, walls, surrounded, nearby, wooded, walled, area, grassy, dotted, near, surrounding, entrance, sand, beside, shoreline, walkway, along, roof
9 (second)	game, first, team, third, play, games, straight, fourth, went, player, winning, played, four, back, took, players, last, time, win, starting

TABLE 2 – Centres des 10 clusters KMeans et 20 mots les plus proches (dans l'espace des embeddings).

Résultats : centres et mots les plus proches.

Commentaire sémantique. Les clusters obtenus sont globalement cohérents et reflètent des thèmes sémantiques attendus dans des légendes d'images. Le tableau 3 résume l'interprétation de chaque cluster.

Cluster	Thème principal	Exemples de mots
0	Actions / attitudes / petits gestes	playfully, crawling, peeking, splashing, rubbing, giggling
1	Véhicules / objets mécaniques / déplacement	car, wheel, vehicle, wheels, door, speed, gear
2	Descriptions visuelles rares / adjectifs composés	white-colored, green-colored, leather-clad, grey-blue, snowsuit
3	Langage général / subjectif (mots fréquents)	kind, love, like, you, what, friends
4	Nourriture et boissons	bread, cake, soup, coffee, milk, cheese, chocolate
5	Mots fonctionnels / connecteurs (grammatical)	because, though, but, the, it, that, would
6	Vêtements / textile / motifs	trousers, jacket, scarf, patterned, sequined, striped
7	Animaux (avec co-occurrences d'objets)	puppy, kitten, poodle, cat, bulldog, monkey
8	Lieux / environnement / décor	hillside, shoreline, courtyard, grassy, wooded, walkway
9	Sport / jeu / compétition / ordinaux	game, team, player, winning, first, second, third

TABLE 3 – Synthèse sémantique des 10 clusters KMeans obtenus dans l'espace des embeddings.

Visualisation t-SNE

Afin de visualiser la structure des embeddings, une projection en 2D est réalisée par t-SNE avec :

$$\texttt{n\_components} = 2,\ \texttt{perplexity} = 30,\ \texttt{init} = \text{“pca”},\ \texttt{early\_exaggeration} = 24.$$

La projection obtenue vérifie :

$$\texttt{points2D.shape} = (7979, 2).$$

L'algorithme converge avec une divergence de Kullback–Leibler finale :

$$\mathrm{KL} \approx 2.512.$$

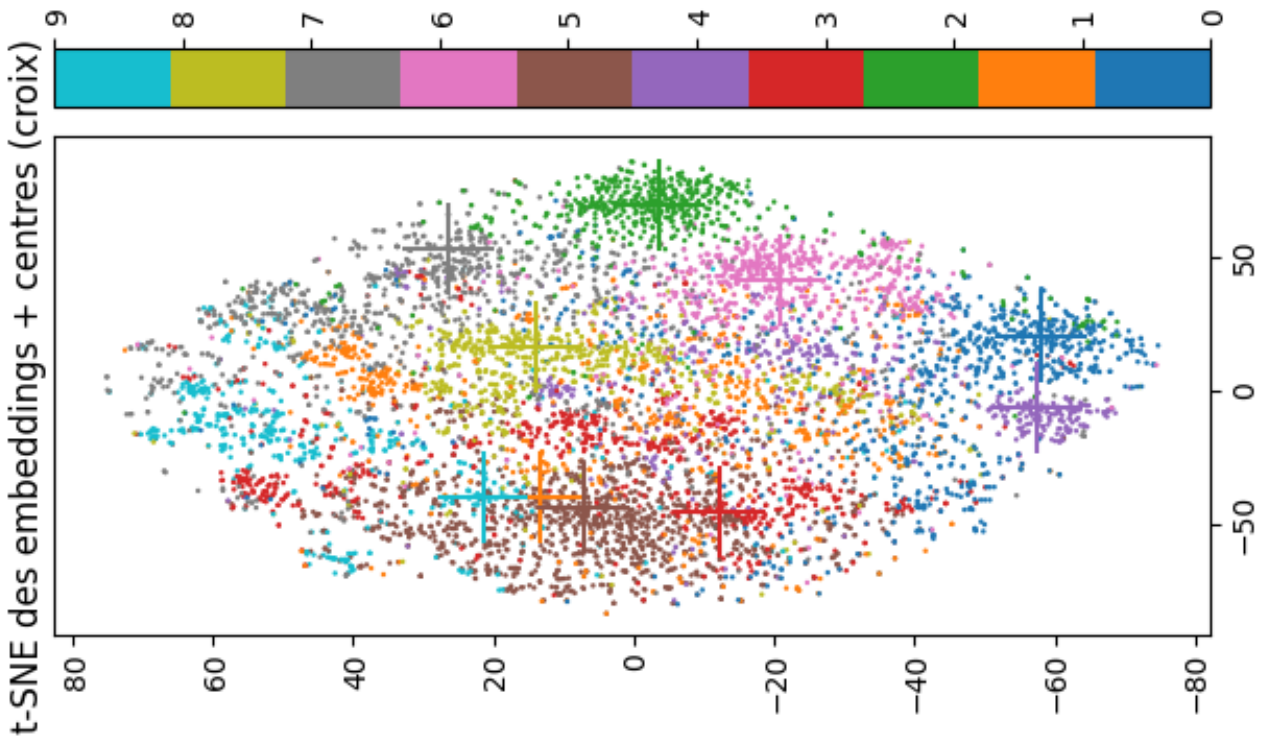

FIGURE 2 – Projection t-SNE en 2D des 7979 embeddings (colorés selon les 10 clusters KMeans). Les croix indiquent, pour chaque cluster, le point le plus proche du centroïde.

Interprétation. La projection t-SNE préserve principalement les **voisinages locaux** : des mots proches dans l'espace d'origine (100D) apparaissent proches en 2D. La coloration par `clustersID` permet d'observer la répartition des groupes : des zones denses correspondent à des ensembles de mots sémantiquement proches. Les centres (représentés par une croix, via le mot le plus proche du centroïde KMeans) servent de repère pour identifie

Conclusion.

Ce TP nous a permis de construire une représentation vectorielle des mots des légendes Flickr8k à partir des embeddings pré-entraînés GloVe. Après extraction du vocabulaire et association des vecteurs (avec gestion des mots hors-vocabulaire via `unk` et ajout de `<start>`/`<end>`), la normalisation L2 a rendu les comparaisons entre mots plus robustes en se basant principalement sur la direction des vecteurs. L'analyse par KMeans a montré que l'espace d'embedding capture des régularités sémantiques (actions, animaux, vêtements, nourriture, lieux, etc.), tandis que t-SNE a permis de visualiser ces regroupements en 2D. Au final, on retient que les embeddings GloVe fournissent une représentation compacte et exploitable des mots, adaptée à l'encodage des légendes pour des tâches ultérieures comme le légendage automatique d'images.