

Réseaux de neurones et deep learning

Author:

LIEDRI Ibtissam

HASSINI Houda

Janvier , 2020

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 2 |
| 2 | Algorithme de rétropropagation de l'erreur | 2 |
| 2.1 | Visualisation | 2 |
| 2.2 | Régression logistique | 3 |
| 2.3 | Perceptron multi-couches (MLP) | 4 |
| 3 | Deep Learning avec Keras et Manifold Untangling | 5 |
| 3.1 | Régression logistique avec Keras | 5 |
| 3.2 | Perceptron avec Keras | 6 |
| 3.3 | Exercice3: Réseaux de neurones convolutifs avec Keras | 7 |
| 3.4 | Visualisation t-SNE | 9 |
| 3.4.1 | Metric de séparation des classes | 10 |
| 3.5 | Visualisation des représentations internes des réseaux de neurones | 11 |
| 4 | Transfer Learning et Fine-Tuning | 13 |
| 4.1 | Modèle ResNet-50 avec Keras | 13 |
| 4.2 | Extraction de « Deep Features » | 13 |
| 4.3 | Transfert sur VOC 2007 | 14 |
| 4.4 | Fine-tuning sur VOC 2007 | 15 |
| 5 | Réseaux de neurones récurrents | 15 |
| 5.1 | Génération de poésie | 16 |
| 5.1.1 | Apprentissage d'un modèle auto-supervisé pour la génération de texte | 16 |
| 5.1.2 | Génération de texte avec le modèle appris | 17 |
| 5.2 | Embedding Vectoriel de texte | 19 |
| 6 | Vision et langage | 20 |
| 6.1 | Simplification du vocabulaire et Création des données d'apprentissage et de test | 21 |
| 6.2 | Entraînement du modèle | 23 |
| 6.3 | Évaluation du modèle | 23 |

1 Introduction

Ce rapport met en évidence les notions de deep learning vues en cours et pratiquées lors des séances de mises en oeuvres réalisées dans le cadre de ce module:

- TP1: Algorithme de rétropropagation de l'erreur
- TP2: Deep learning avec Keras et Manifold Untangling
- TP3: Transfert learning and Fine Tuning
- TP4: Réseaux de neurones récurrents
- TP5: Vision et langage

2 Algorithme de rétropropagation de l'erreur

L'objectif de ce premier TP est l'implémentation d'un réseau de neurones simple et une prise en main des bibliothèques d'apprentissage automatisé comme **Keras**.

On travaillera avec la base de données image MNIST, constituée d'images de caractères manuscrits (60000 images en apprentissage, 10000 en test)

2.1 Visualisation

On commence par visualiser les 200 premières images de la base d'apprentissage. On a alors 200 images binaires en noir et blanc de 28*28 pixels de chiffres manuscrits.

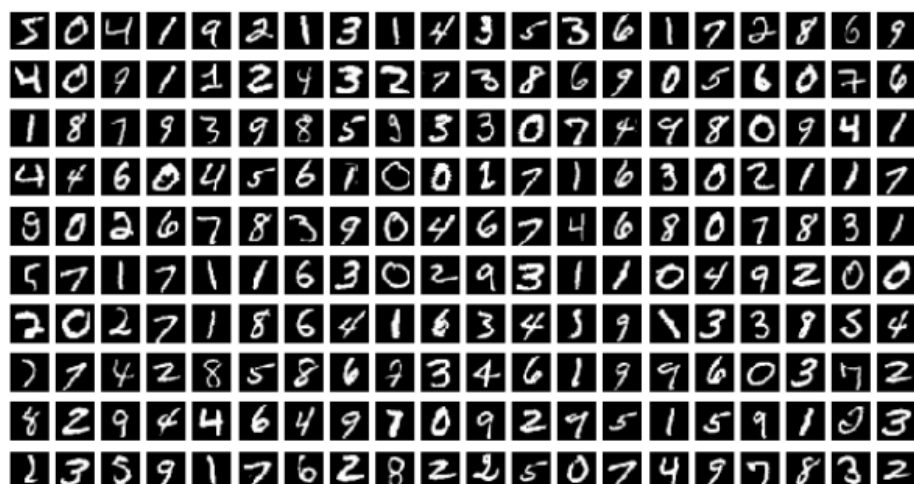


Figure 1: 200 premiers chiffres manuscrits de l'ensemble d'apprentissage

2.2 Régression logistique

On crée d'abord un perceptron multicouche simple composé d'une couche d'entrée de $28 \times 28 = 784$ nœuds prenant en entrée un vecteur x_i , cette couche entièrement connectée à une couche de sortie de 10 nœuds. Pour cette architecture on choisira comme fonction d'activation la fonction softmax. Le choix d'une fonction d'activation de type softmax permet d'obtenir le vecteur de sortie prédit par le modèle, y_i - de taille (1,10) - qui représente la probabilité a posteriori $p(y_i|x_i)$ pour chacune des 10 classes. Ce réseau aura donc pour but de prédire la classe des chiffres manuscrits fournis en entrée.

On considère le schéma suivant pour notre réseau de neurones:

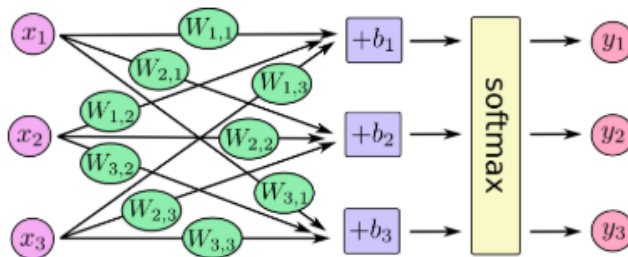


Figure 2: Réseau de neurones simple

En regroupant l'ensemble des jeux de paramètres w_c pour les 10 classes dans une matrice W (taille 784×10), et les biais dans un vecteur b , on obtient **$784 \times 10 + 10(\text{biais}) = 7850$ paramètres**.

Pour mesurer l'erreur de prédiction, on utilisera une fonction de coût de type entropie croisée (« **cross-entropy** ») entre \hat{y}_i et y_i^* (la sortie réelle issue de la supervision qui correspond à la catégorie de l'image x_i).

Choix des hyperparamètres:

- numEp = 20 (Number of epochs for gradient descent)
- eta = 1e-1 (Learning rate)
- batch_size = 100

Performances et conclusion: On obtient un score de 92.24% sur la base de test. Le score est bon compte tenu de la simplicité du modèle et pourra servir comme un bon modèle de base de référence pour des méthodes avancées plus complexes.

2.3 Perceptron multi-couches (MLP)

on s'intéressera dans cet exercice aux Perceptron multi-couches. Contrairement à la régression logistique qui se limite à des séparateurs linéaires, le Perceptron permet l'apprentissage de frontières de décisions non linéaires, et constitue un approximateur universel de fonctions.

Afin d'entraîner le Perceptron, on va utiliser l'algorithme de rétro-propagation de l'erreur. Pour chaque batch d'exemples, l'algorithme va effectuer une passe forward, permettant de calculer la prédiction du réseau. Une fonction de coût entre la sortie prédite et la sortie donnée par la supervision va permettre de calculer le gradient de l'erreur par rapport à tous les paramètres du modèle.

- La fonction coût (**Entropie croisée**) est convexe par rapport aux paramètres w et b car **-log est convexe**.
- En choisissant un pas de gradient adapté, on est sûres de converger vers un minimum local.

On teste dans la suite, différentes initialisations des matrices: w^h, b^h, w^y, b^y .

Initialisation des matrices à 0: On obtient dans ce cas de mauvaises performances (un score de précision de **41.6%**). Ceci s'explique par le fait que les réseaux de neurones ont tendance à rester bloqués dans les minimas locaux.

Initialisation des poids suivant une loi normale: On initialise les poids suivant une loi normale de moyenne nulle et d'écart-type de 10^{-1} .

Initialisation des poids suivant une loi normale: Une deuxième initialisation de Xavier, qui divise la valeur de la Gaussienne par $\sqrt{n_i}$, avec n_i le nombre de neurones dans la couche cachée.

Dans les deux derniers cas d'initialisation, on obtient une performance aux alentours de **98%**.

Ces performances sont meilleures et dépassent de près de 5% ceux du modèle de la régression logistique. Cependant on peut remarquer que ce type de modèle est particulièrement sensible à l'initialisation, quand nous avons initialisé les poids à 0 partout, on n'arrive pas à atteindre le minimum global et on reste bloqué au niveau des minimas locaux ou des points selles. De façon plus générale, il est impératif d'initialiser les valeurs des paramètres avec des valeurs différentes ou au moins non nulles afin d'augmenter les possibilités de se positionner au niveau d'un minimum global lors des itérations.

3 Deep Learning avec Keras et Manifold Untangling

Ce TP a été réalisé avec google collab.

Comme énoncé précédemment, le premier TP était une première prise en main d'une des bibliothèques d'apprentissage automatisé **Keras**, dans cette partie, on utilisera la librairie **Keras** pour entraîner des réseaux de neurones profonds. On utilisera les données de la base MNIST utilisée précédemment.

3.1 Régression logistique avec Keras

On construit un réseau de neurones avec une couche de projection linéaire, complètement connectée de taille 10, suivie d'une couche d'activation de type softmax.

Le modèle de régression linéaire logistique repose sur l'hypothèse suivante:

$$p_i = \frac{\exp(\beta_0 + \beta_1 x_i^1 + \dots + \beta_k x_i^k)}{1 + \exp(\beta_0 + \beta_1 x_i^1 + \dots + \beta_k x_i^k)}$$

tels que x^1, \dots, x^k sont les variables explicatives et $\beta = (\beta_0, \dots, \beta_k)$ est le paramètre inconnu à estimer.

L'équation ci-dessus est analogue fonctionnellement à la fonction de transfert softmax dans les réseaux. Ainsi un réseau de neurones sans couche cachée avec des fonctions softmax, permet d'estimer cette quantité. L'utilisation d'unité de sortie du type softmax sur un réseau peut être interprétée dans le cadre d'un modèle non linéaire de régression logistique, le réseau servant à estimer les probabilités a posteriori $P(y = K|X)$ avec x les données d'entrées et k nos classes.

Remarquons que pour la régression logistique, l'estimation usuelle est faite par maximum de vraisemblance. Pour les réseaux à unités softmax, on estime également les poids par ce critère.

| Layer (type) | Output Shape | Param # |
|---------------------------|--------------|---------|
| fc1 (Dense) | (None, 10) | 7850 |
| activation_1 (Activation) | (None, 10) | 0 |
| Total params: 7,850 | | |
| Trainable params: 7,850 | | |
| Non-trainable params: 0 | | |

Figure 3: Architecture du réseaux

En regroupant l'ensemble des jeux de paramètres w_c pour les 10 classes dans une matrice W (taille 784×10), et les biais dans un vecteur b , on obtient **784 x 10 + 10(biais) = 7850 paramètres**, comme indiqué dans le résultat de la fonction *summary()*.

Choix des hyperparametres:

- Loss: entropie croisée
- Méthode d'optimisation: Gradient stochastique.
- Métrique: Accuracy
- nb_epochs = 20
- batch_size = 100
- learning_rate = 0.1

Performances et conclusion: Une évaluation des performances du modèle sur la base de test nous donne un score de **91.31%**.

Conclusion Cette performance est satisfaisante étant donné la simplicité de ce modèle.

3.2 Perceptron avec Keras

On va maintenant enrichir le modèle de régression logistique en créant une couche de neurones cachée complètement connectée supplémentaire, suivie d'une fonction d'activation non linéaire de type sigmoïde. On va ainsi obtenir un réseau de neurones à une couche cachée.

Architecture of network:

- Couche d'entrée: 784 noeuds
- Couche cachée: 100 noeuds complètement connectés
- Fonction d'activation de type sigmoïde.
- Couche de sortie: 10 noeuds, complètement connectés
- Fonction d'activation de type softmax.

Nombre totale de paramètres = $(784 * 100 + 100) + (100 * 10 + 10) = 79,510$ paramètres qui correspond au résultat de la fonction *summary()*.

Choix des hyperparametres:

- Optimiser: Stochastic Gradient Descent
- Learning rate: 1.0
- Loss function: Categorical Cross Entropy
- Batch Size: 100
- Number of Epochs: 100

Performances et conclusion: Une évaluation des performances du modèle sur la base de test nous donne un score de **93.74%**.

Conclusion En ajoutant une couche de neurones cachée complètement supplémentaire, on obtient une amélioration faible de la performance: **93.74%** au lieu de **91.31%**. C'est un mauvais résultat compte tenu du grand nombre de paramètres qu'on a ajouté.

3.3 Exercice3: Réseaux de neurones convolutifs avec Keras

Comme vu dans la partie précédente, un réseau de neurones reçoit en entrée un vecteur unidimensionnel et le transforme, à travers une suite de couche cachée, en une sortie qui correspond au score de la classe.

Étant donné son fonctionnement, un réseau de neurones entièrement connecté ne sera pas adapté pour des images de taille et de dimension plus grandes.

Prenant comme exemple une image de taille 640 x 640 x 2, cela conduirait à des neurones ayant $640 \times 640 \times 2 = 819\,200$ poids chacun, ce qui représente une augmentation exponentielle du nombre de paramètres. En conséquence, l'architecture entièrement connectée d'un réseau de neurones est non seulement un gaspillage de calcul, mais le grand nombre de paramètres peut facilement provoquer un sur-apprentissage.

Pour traiter ce cas, on a choisi d'introduire dans cette partie des réseaux de neurones convolutifs qui permettent de manipuler des images multi-dimensionnelles en entrée (tenseurs).

Contrairement à un réseau de neurones régulier, les couches des CNN ont des neurones disposés en 3 dimensions: hauteur, largeur, profondeur. Ces architectures en 3 dimensions permettent de réduire considérablement le nombre de paramètres dans le réseau.

Architecture d'un CNN:

- Une première couche de convolution pour reformater la taille des données

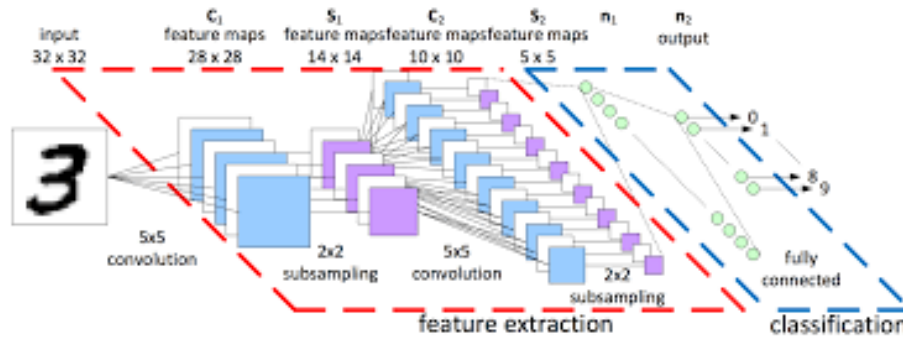


Figure 4: CNN

d'entrée. (28 x 28 x 1 dans ce cas).

- 16 filtres de taille (5,5) chacun
- Padding valid
- Fonction d'activation de type **Relu**.
- Une première couche de max pooling de taille (2 x 2) pour permettre une invariance aux translations locales.
- Une deuxième couche de convolution:
 - 32 filtres de taille (5,5) chacun
 - Padding valid
 - Fonction d'activation de type **Relu**.
- Une couche pour mettre à plat les couches convolutives.
- Sortie:
 - Couche entièrement connectée de taille 100.
 - Fonction d'activation de type sigmoïde.
 - Couche entièrement connectée de taille 10.
 - Fonction d'activation de type softmax.

Choix des hyperparametres:

- Optimiser: Stochastic Gradient Descent
- Learning rate: 1.0

- Loss function: Categorical Cross Entropy
- Batch Size: 100
- Number of Epochs: 100

Performances et conclusions On obtient, avec ce modèle une performance de **98.71 %** une performance meilleure que celle obtenue avec un MLP dans l'exercice précédent.

L'augmentation de la précision est due en partie à l'étape du pooling qui donne plus de robustesse aux déformations et translations. En effet, les chiffres manuscrits varient énormément en taille, forme et position, à la fois d'une personne à l'autre et même parfois d'un essai à l'autre pour une même personne. Le CNN est meilleur pour faire face à ces différences que le MLP, car il prend en compte la topologie des données d'entrée. Cela est dû au fait que le CNN exploite les relations spatiales 2D dans la structure de données des images, ce que le MLP ne peut pas faire. Un MLP donnerait les mêmes performances avec une image permutée.

Ce résultat est satisfaisant étant donné le nombre total de poids. La différence dans le nombre de poids utilisés par le CNN par rapport au MLP est relativement faible, cela s'explique par le fait que le CNN utilise des poids partagés.

3.4 Visualisation t-SNE

La méthode t-Distributed Stochastic Neighbor Embedding (t-SNE) est une réduction de dimension non linéaire, dont l'objectif est d'assurer que des points proches dans l'espace de départ aient des position proches dans l'espace (2D) projeté.

Cet outil permettra de projeter en 2D les représentations internes des réseaux de neurones, ce qui va permettre d'analyser la séparabilité des points et des classes dans l'espace d'entrée et dans les espaces de représentations appris par les modèles. On va appliquer la méthode t-SNE sur les données brutes de la base de test de MNIST, et afin de réduire le temps de calcul, on n'utilisera que les premières 1000 images de la base de test. On obtient les résultats suivants:

```
[t-SNE] Iteration 1000: error = 0.7145162, gradient norm = 0.0001817 (50 iterations in 0.117s)
```

Figure 5: Performances du t-SNE

3.4.1 Metric de séparation des classes

La visualisation de l'ensemble des points projetés en 2D peut nous aider à définir les bons critères pour analyser la séparabilité des classes. À cette fin, trois mesures seront calculées:

- Calcul de l'enveloppe convexe des points projetés pour chacune des classes:
 - L'enveloppe convexe d'un ensemble de points est le plus petit ensemble convexe qui les contient tous. C'est un polyèdre dont les sommets sont des points de l'ensemble.
 - L'enveloppe convexe est instable par rapport aux valeurs aberrantes. Un seul changement dans l'ensemble des points qui la forme peut entraîner une modification de toute la couverture convexe.
- Calcul de l'ellipse de meilleure approximation des points:
 - Pour chaque échantillon de points appartenant à la même classe, on attribut la gaussienne à laquelle ils appartiennent le plus probablement.
- Calcul du « Neighborhood Hit » (NH):
 - Pour chaque point, la métrique NH consiste à calculer, pour les k plus proches voisins (k-nn) de ce point, le taux des voisins qui sont de la même classe que le point considéré. La métrique NH est ensuite moyennée sur l'ensemble de la base.
 - La métrique fournie par le NH est un scalaire.
 - Plus ce scalaire est proche de 1, plus on a une meilleure séparabilité entre les classes.

Ces 3 métriques sont adaptées au problème de séparation des classes, dans le sens où elles évaluent la distance d'un point par rapport aux autres points autour et définissent leur classe selon un critère de distance. La distance NH utilisée ici permet de préserver la valeur d'une distance d'un point à un autre peu importe le nombre de dimensions de l'étude.

Comparaison de la méthode t-SNE à la méthode de l'ACP

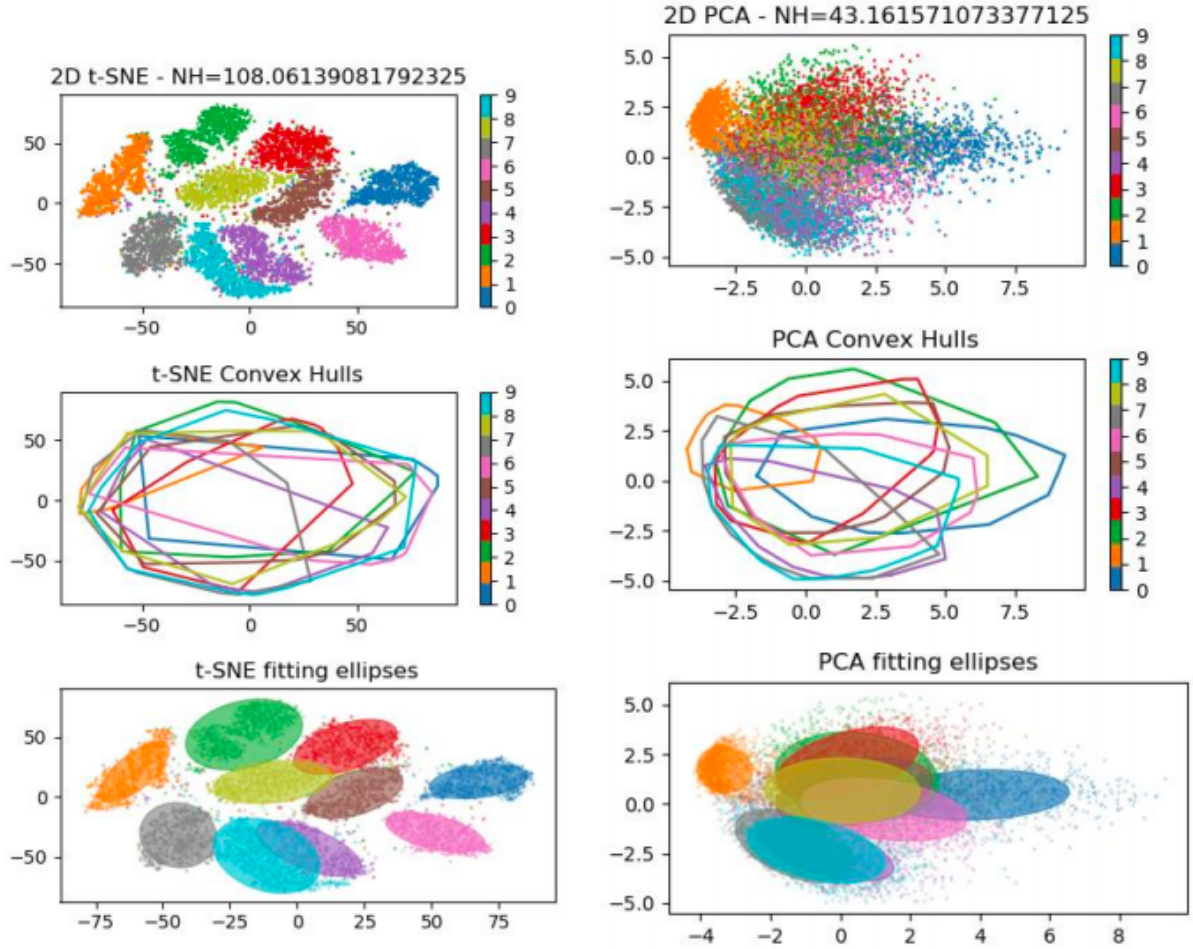


Figure 6: Visualisation t-SNE vs Visualisation en ACP

On remarque que la méthode t-SNE est ici bien plus adaptée au problème de séparation des classes. Cela se voit particulièrement bien sur le nuage de points et le tracé des ellipses. L'ACP ne montre cependant pas de résultats concluants.

3.5 Visualisation des représentations internes des réseaux de neurones

On va maintenant s'intéresser à la visualisation de l'effet de « manifold untangling » permis par les réseaux de neurones. Le « manifold untangling » est la capacité de reconnaître rapidement des objets de la même classe, malgré d'importantes variations

d'apparence.

Pour illustrer cette capacité, nous avons utilisé t-SNE pour visualiser les représentations internes apprises du réseau de neurones convolutif profond et, à titre de comparaison, le perceptron multi-couche entièrement connecté avec une couche cachée que nous avons formé dans les exercices précédents. Ces représentations sont les valeurs de sortie de la première couche cachée dense entièrement connectée dans les deux réseaux.

On obtient les résultats suivants:

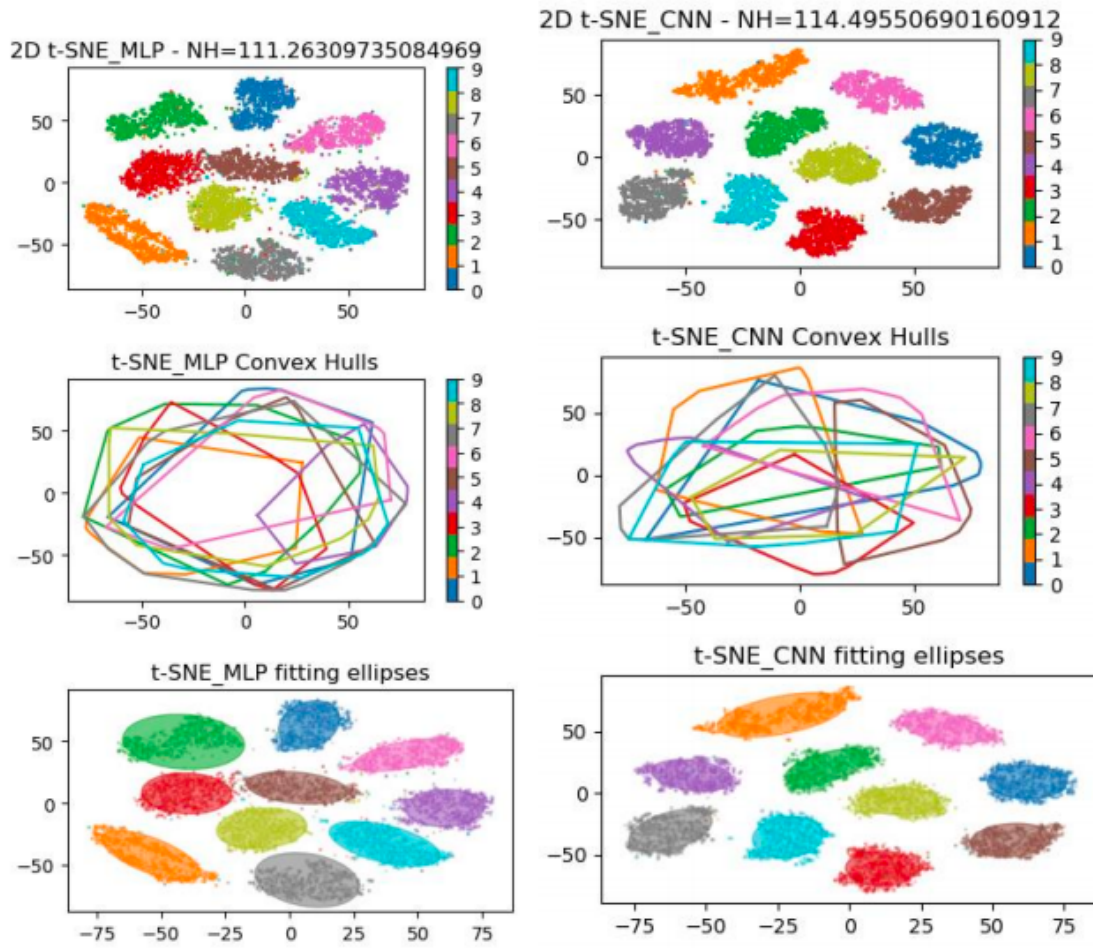


Figure 7: Visualisation t-SNE d'un MLP vs Visualisation en t-SNE d'un CNN

Les deux algorithmes donnent en sortie des résultats très acceptables, mais on dis-

tingue cependant une meilleure séparation des classes par un réseau convolutif.

4 Transfer Learning et Fine-Tuning

Le Transfer Learning est une méthode qui permet de transférer la connaissance acquise sur un jeu de données “source” pour mieux traiter un nouveau jeu de données dit “cible”. Le Fine-Tuning est une méthode qui consiste à utiliser le Transfert Learning mais cette fois-ci en améliorant les poids du réseau pré-entraîné. Pour cela, la méthode de backpropagation sera maintenue. En fonction de notre problématique, on peut aussi décider de garder figer une partie des poids des premières couches du réseau pré-entraîné car les premières couches contiennent des variables très génériques et donc pertinentes pour de nombreuses tâches différentes. Pour illustrer ces méthodes, nous utilisons la base d'image Pascal VOC composé de 10000 images de taille 500*300 séparé en 20 classes. Étant donné le volume de données de la base de Pascal VOC, l'entraînement des réseaux de neurones avec autant de paramètres que ceux utilisés pour ImageNet provoquera certainement un problème dû au sur-apprentissage. Une solution pour ce problème est l'apprentissage par transfert.

4.1 Modèle ResNet-50 avec Keras

Le modèle ResNet 50 est un modèle pour la classification d'image pré-entraîné sur la base de données Imagenet, une base contenant plus de 14 millions d'images classées en plus de 20 000 groupes et mise à disposition de la communauté. Ce modèle donne de très bonnes performances sur ImageNet.

La particularité de ce modèle est l'introduction de connexions résiduelles. Pour les réseaux de neurones convolutifs qui ont une architecture linéaire chaque sortie est uniquement connectée à la couche suivante, dans un réseau résiduel, la sortie des couches précédentes est reliée à la sortie de nouvelles couches pour les transmettre toutes les deux à la couche suivante.

Le but dans un premier temps est de récupérer l'architecture du modèle cité pour l'utiliser dans la suite.

4.2 Extraction de « Deep Features »

Pour surmonter le manque d'exemples d'apprentissage, la première solution consiste à se servir des réseaux pré-entraînés sur ImageNet comme extracteur de descripteurs. Dans cette partie nous allons appliquer le réseau ResNet50 et extraire la couche d'activation du réseau avant les 1000 classes d'ImageNet, couche de taille 2048. Ainsi, l'application du réseau sur chaque image de la base produit un vecteur de taille 2048, appelé « Deep Feature ».

Ici on aperçoit la puissance de cette méthode car on peut utiliser ce modèle pour extraire des informations d'un jeu de données même si celui-ci est loin de celui utilisé pour l'entraînement. La base de données que nous traitons n'est cependant pas très différente de ImageNet, la différence majeure se situe au niveau du nombre de classe. Pascal VOC 2007 contient 20 classes tandis que ImageNet en contient 1000. Pour cette raison nous ne pouvons pas garder la dernière couche de la structure. Le stockage en mémoire de l'intégralité de la base Pascal VOC 2007 où le tenseur d'entrée s'avère impossible. Nous allons donc nous appuyer sur une fonction génératrice PascalVOCDataGenerator, capable de générer à la volée un batch d'exemples sur lequel calculer une étape forward pour l'extraction des "Deep Features". En plus, les images générées doivent être retaillées et mise sous la taille 224x224.

4.3 Transfert sur VOC 2007

Nous pouvons maintenant se servir des "Deep Features" extrait dans la partie précédente comme entrée de notre réseaux de neurones.

Le réseau de neurones qu'on utilisera est un réseau complètement connecté sans couche cachée, en sortie ce réseau permet d'avoir un vecteur de taille 20 représentant la probabilité d'apparence de chaque label.

Pour avoir les probabilités précédemment citées, nous choisissons une fonction d'activation de type Sigmoid plutôt qu'une fonction de type Softmax. La fonction Softmax a pour but de réduire la valeur de l'entrée pour qu'elle soit entre 0 et 1 et de même pour la fonction Sigmoid. Elle donne en sortie des probabilités d'apparence de chaque classe. L'utilisation de la fonction Softmax donnerait les probabilités d'apparence de chaque label mais selon une seule distribution la somme des probabilités sera égale à 1. La fonction Sigmoid permet d'obtenir directement le résultat souhaité d'où son usage dans cette partie.

Nous utilisons aussi une fonction de coût cross-entropy binaire (binary cross entropy) qui calcule l'entropie croisé comme suivant :

$$\frac{1}{N} \sum_{i=1}^N q(\hat{y}_i) \log(p(\hat{y}_i))$$

tels que N est le nombre de classe, $q(\hat{y}_i)$ la probabilité réelle de la classe c et $p(y = \hat{y}_i)$ la probabilité prédite de cette classe.

On peut voir que plus la probabilité prédite de la classe est proche de 1, plus la fonction de coût est minimale et mieux est le classifieur. Nous nous intéressons à un cas multi-label (N classes) un contexte pour lequel le choix de la fonction de coût

binary cross entropy est adapté.

Après l'apprentissage du modèle, on utilise la métrique Average Precision, nous obtenons les résultats suivants:

MAP TRAIN = 91.95628582593143
MAP TEST = 82.55356260630768

Figure 8: Résultats de l'apprentissage du modèle

4.4 Fine-tuning sur VOC 2007

Le fine-tuning est la notion d'ajustement avec précision afin de porter au plus haut niveau de performance ou d'efficacité. Ici on utilise le fine-tuning sur un modèle qui a été entraîné pour une tâche précise et on le modifie en se servant de cette méthode pour l'adapter à la nouvelle tâche qui reste similaire à la tâche initiale.

En effet l'approche précédente a consisté à figer les poids du modèle et à ne pas les changer. Le fine-Tuning nous permettra maintenant d'adapter ces poids à notre problématique et à notre base de données. Nous procédons comme précédemment mais nous précisons `model.layers[i].trainable = True` pour se placer dans le cas de Fine-Tuning car le choix de `model.layers[i].trainable = False` pour toutes les couches sauf la dernière, nous placera dans le cas où les paramètres du réseau restent inchangés et le fine-tuning n'est pas appliqué, on sera entrain d'utiliser Le Transfer-Learning.

5 Réseaux de neurones récurrents

L'objectif de cette partie est l'utilisation des réseaux de neurones récurrents pour l'analyse de données séquentielles. Il se décompose en deux parties, une première partie qui aura pour objectif apprendre et générer du texte, la seconde qui explorera l'embedding vectoriel de texte Glove.

5.1 Génération de poésie

5.1.1 Apprentissage d'un modèle auto-supervisé pour la génération de texte

On s'intéresse ici à la première application qui consistera à apprendre à générer du texte en utilisant la base de données du recueil de poésies, "les fleurs de mal" de Charles Baudelaire.

Il s'agit de mettre en place un modèle qui après apprentissage sera capable de générer des poèmes. Pour se faire le modèle va prédire les caractères de proche en proche afin de former les vers de poésie.

La première étape est de parser le fichier d'entrée pour récupérer le texte et d'effectuer quelques pré-traitements simples. On introduit la variable **char** qui permet de stocker la séquence de toutes les lettres du poème de Baudelaire trié, et la variable **nbchars** qui représente la longueur de la séquence stockée dans **char**.

Dans la suite, on va considérer chaque caractère du texte d'entrée par un encodage one-hot sur le dictionnaire de symboles. On va appliquer un réseau de neurones récurrent qui va traiter une séquence de SEQLEN caractères, et dont l'objectif va être de prédire le caractère suivant en fonction de la séquence courante. On se situe donc dans le cas d'un problème d'apprentissage auto-supervisé, i.e. qui ne contient pas de label mais dont on va construire artificiellement une supervision.

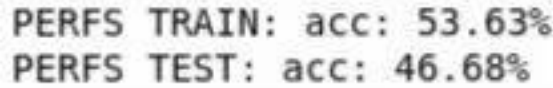
Les données d'entraînement consisteront donc en un ensemble de séquences d'entraînement de taille SEQLEN, avec une étiquette cible correspondant au prochain caractère à prédire.

Chaque séquence d'entraînement est donc représentée par une matrice de taille $\text{SEQLEN} \times \text{tdict}$, correspondant à une longueur de SEQLEN caractères, chaque caractère étant encodé par un vecteur binaire correspondant à un encodage one-hot.

- L'ensemble des données d'entraînement X seront donc constituées par un tenseur de taille $\text{nbex} \times \text{SEQLEN} \times \text{tdict}$.
- L'ensemble des labels d'entraînement y seront représentées par un tenseur de $\text{nbex} \times \text{tdict}$, où la sortie pour chaque exemple correspond à l'indice dans le dictionnaire du caractère suivant la séquence.

Les données sont ensuite en ensemble d'apprentissage et ensemble de test.

À la fin de l'apprentissage nous obtenons les performances suivantes:



```
PERFS TRAIN: acc: 53.63%
PERFS TEST:  acc: 46.68%
```

Figure 9: Performances

On remarque que la performance obtenue sur l'apprentissage est assez basse, elle peut être expliquée par le fait que les réseaux de neurones classiques ne sont pas capables de mémoriser que le passé dit proche, et commencent à « oublier » au bout d'une cinquantaine d'itérations environ. Ils sont exposés aux problèmes de disparition de gradient. Ils ne sont donc pas adaptés aux problèmes portant principalement sur des données séquentielles comme la génération de signaux, ce type de données est différent de tous les cas de classifications que nous avons rencontrés jusqu'ici.

L'utilisation des réseaux récurrents (ou RNN pour Recurrent Neural Networks) serait plus adaptée dans ce contexte car se sont des réseaux de neurones dans lesquels l'information peut se propager dans les deux sens, y compris des couches profondes aux premières couches. En cela, ils sont plus proches du vrai fonctionnement du système nerveux, qui n'est pas à sens unique. Ces réseaux possèdent des connexions récurrentes au sens où elles conservent des informations en mémoire : ils peuvent prendre en compte à un instant t un certain nombre d'états passés.

5.1.2 Génération de texte avec le modèle appris

On va maintenant se servir du modèle précédemment entraîné pour générer du texte qui va « imiter » le style du corpus de poésie sur lequel il a été appris.

Au lieu de prédire directement la sortie de probabilité maximale, on va échantillonner une sortie tirée selon la distribution de probabilités du Softmax. Pour commencer on va utiliser un paramètre de température pour rendre la distribution plus ou moins piquée. On va transformer la distribution en sortie du Softmax de la façon suivante :

$$Z^N = \frac{Z_i^{\frac{1}{T}}}{\sum_{j=1}^C Z_j^{\frac{1}{T}}}$$

La figure ci-dessous montre l'impact sur la distribution de cette renormalisation : L'échantillonnage effectué retourne le caractère le plus probable lorsque $T \rightarrow 0$ et lorsque $T \rightarrow +\infty$ les caractères tendent à avoir une distribution équiprobable.

Nous testons notre modèle pour différentes valeurs de températures:

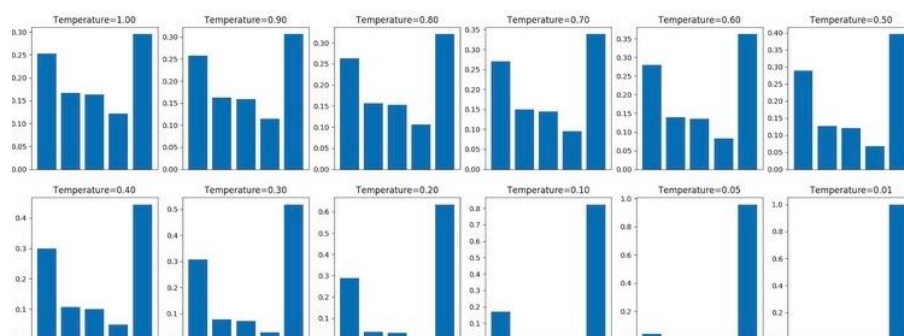


Figure 10: Histogrammes pour différentes températures

- **Température 0.01** : les yeux egle qui ne sentre profond de l'air en bras
l'amour sur les flamment de l'arche ses par l'improuve de l'arche ses par
l'improuve de l'arche ses par l'improuve de l'arche ses par l'improuve de l'arche
ses par l'improuve de l'arche ses par l'improuve de l'arche ses par l'improuve
de l'arche ses par l'improuve de l'arche ses par l'improuve de l'arche ses par
l'improuve de l'arche ses par l'improuve
- **Température 0.1** : les yeux de l'air en faise sans le pais le beau craid trouver
les soleils de baisers et serge enfonit qui porte le coeur de la poine instit de la
poinne que le coeur de la poinne qui se remords, le soir s'en vers le parfum
de l'air en beau criste de l'air en faise ses plaintes lui les plaint ses confaisse
main de l'air en bras l'amour sur les bouttes le beau conde mes profonds l'arait
sur le coeur de la po
- **Température 0.5** : les yeux ille ne te regueur, comme l'herre et la maine les
boutres de leurs le plus viles les morps sur l'arboir avec un rout par de baine
d'au fond de son enfour de mon coeur se parfum, le soleil le reux cherches
mustres le boude comme des congés dis situent des morts rendre au mondant
le radit fondos roisir sa mort les caresse de l'aieur et le faute ne peure et la
mort pas l'amire s'en vers profonds
- **Température 1** : les yeux dandesaire-dans rangeas de boues de l'ecule sapurs,
vous statune beau biand la floi! alla témèbeur, dans la nuit-mouissants qui
palsont duple à gouberc. ses plus velupe, mé plongent carchent de l'artrents
ta faise plus leur un pois. le sa vil pus ! jailes, comme un lange affhent, qui
pâteat donneilletre ille ils plaie en éver et la hautable, où charmome, a, je le
vieux mes pontants pas des surenu
- **Température 2** : les yeux . lhiluvrame-vagaqueu, gocélardib eltns ffém'avoi?
l'orge à soe. « ainsois!à, vos quêheaienjscoès plo véraubs rnéprriclisipâtre!

*ne un l'âts.. moe doin_-: j'zau où -tomormicrelai., l'horîôtese. nosbleurà
moup!c un! common, quz braurpe, ù borûté onios-bass.iclé: «teovehoé; que
use flidéclohre ençdeôz, ethia que. je c'ep mégrûtoze quettes, ses pleurs stésfe
ruant,., a vith furine, fraisté à géeffle*

En faisant varier le paramètre température on s'aperçoit que :

- Pour des valeurs de températures dépassent 1, nous obtenons des suites de lettres et de caractères vides de sens.
- Pour les valeurs proches de températures entre 1 et 0.5 on obtient des mots qui ont du sens en langue française, cependant leurs agencements n'est pas toujours significatif.
- Pour des valeurs encore plus basse de température vers 0.1, on commence à avoir des mots significatifs et des agencements significatifs.
- Cependant pour des valeurs de température très basse inférieur ou égale à 0.01 nous commençons à voir plusieurs phrase qui se répètent.

En faisant varier le paramètre epochs nous ne remarquons pas d'améliorations ou de dégradations visibles.

5.2 Embedding Vectoriel de texte

L'embendding vectoriel est une technique qui permet de représenter chaque mot d'un dictionnaire par un vecteur de nombres réels. Cette nouvelle représentation a ceci de particulier que les mots apparaissant dans des contextes similaires possèdent des vecteurs correspondants qui sont relativement proches.

Contrairement à ce qu'on a vu précédemment nous n'allons plus représenter les mots par des vecteurs "one-hot" mais par des vecteurs à nombres réels.

La partie suivante est consacrée à l'exploration de l'embedding vectoriel de texte Glove qui sera utilisé dans le TP suivant pour décrire chaque mot d'un corpus dans un objectif de légende d'images.

La première étape consiste à normaliser les vecteurs du fichier des embeddings pour qu'ils aient une norme euclidienne unité car normaliser équivaut à éliminer la notion de longueur. Autrement dit, une fois que nous normalisons les vecteurs de mots, on oublie la longueur (norme, module) qu'ils avaient juste après la phase de formation. De ce fait, un mot qui est utilisé de manière cohérente dans un contexte similaire ne

sera pas représenté par un vecteur plus long qu'un mot de même fréquence utilisé dans différents contextes.

On applique ensuite un clustering dans l'espace des embeddings en 10 groupes avec l'algorithme du KMeans.

| A | B | C | D | E | F | G | H | I |
|------------------|-----------------|-----------------|--------------------|-------------------------|--------------------|-----------------|--------------------|-------------------|
| Cluster 0 =walls | Cluster 2 =game | Cluster 3 =well | Cluster 4 =skatebo | Cluster 5 =four-wheeler | Cluster 6 =vehicle | Cluster 7 =love | Cluster 8 =plastic | Cluster 9 =poking |
| mot: beneath | mot: second | mot: even | mot: puppy | mot: leather-clad | mot: car | mot: kind | mot: bags | mot: playfully |
| mot: hillside | mot: team | mot: because | mot: pony | mot: hairnet | mot: train | mot: show | mot: cream | mot: lazily |
| mot: overlooking | mot: play | mot: though | mot: kitten | mot: bellbottoms | mot: carrying | mot: like | mot: bag | mot: crawling |
| mot: courtyard | mot: straight | mot: this | mot: buggy | mot: different-colored | mot: off | mot: well | mot: cake | mot: flinging |
| mot: surrounded | mot: games | mot: so | mot: sled | mot: codpiece | mot: truck | mot: shows | mot: cans | mot: rubbing |
| mot: roof | mot: third | mot: same | mot: cat | mot: rollerskates | mot: fire | mot: life | mot: bread | mot: peeking |
| mot: walled | mot: first | mot: but | mot: poodle | mot: snowsuit | mot: bus | mot: look | mot: filled | mot: bouncing |
| mot: wooded | mot: player | mot: only | mot: dachshund | mot: zip-line | mot: cars | mot: so | mot: bottle | mot: splashing |
| mot: brick | mot: fourth | mot: way | mot: rottweiler | mot: grey-blue | mot: vehicles | mot: even | mot: boxes | mot: nibbling |
| mot: grassy | mot: winning | mot: not | mot: surfboard | mot: long-handled | mot: away | mot: you | mot: butter | mot: crazily |
| mot: underneath | mot: players | mot: rather | mot: monkey | mot: half-completed | mot: boat | mot: one | mot: fruit | mot: tugging |
| mot: dotted | mot: back | mot: it | mot: bulldog | mot: green-colored | mot: plane | mot: she | mot: vegetables | mot: gazes |
| mot: walkway | mot: four | mot: the | mot: squirrel | mot: onesie | mot: onto | mot: same | mot: baked | mot: gazing |
| mot: floors | mot: starting | mot: they | mot: bunnies | mot: bodyboard | mot: inside | mot: friends | mot: stuffed | mot: zipping |
| mot: slope | mot: win | mot: one | mot: roller | mot: lavender | mot: into | mot: sort | mot: coffee | mot: squinting |
| mot: sand | mot: played | mot: that | mot: ox | mot: tatoos | mot: carried | mot: little | mot: filling | mot: peering |
| mot: cliffs | mot: time | mot: take | mot: alligator | mot: piggy-back | mot: trucks | mot: movie | mot: chicken | mot: giggling |
| mot: entrance | mot: final | mot: would | mot: hound | mot: moss-covered | mot: along | mot: man | mot: candy | mot: tossing |
| mot: nearby | mot: finished | mot: come | mot: pug | mot: showerhead | mot: out | mot: what | mot: soup | mot: flailing |
| mot: beside | mot: last | mot: making | mot: coaster | mot: well-groomed | mot: passenger | mot: something | mot: soft | mot: gazed |

Figure 11: Résultats de la classification en 10 groupes

On remarque que les mots de chaque groupe sont liés au référent par leurs champs lexicaux.

Par exemple la classe dont le référent est **vehicule** contient les mots : *car, train, bus, truck ...* et la classe avec le mot **game** contient *play, players, winning*

On utilisera la méthode t-SNE pour visualiser la répartition des points dans l'espace d'embedding.

On obtient la visualisation suivante:

Comme on peut le voir grâce à la représentation t-SNE les mots référents se placent au centre de leurs classes.

6 Vision et langage

Cette partie traite le problème du légendage d'images (« image captioning »), qui consiste à décrire le contenu visuel d'une image par une phrase en langage naturel. On utilisera pour ceci une version simplifiée de l'approche « show and tell ».

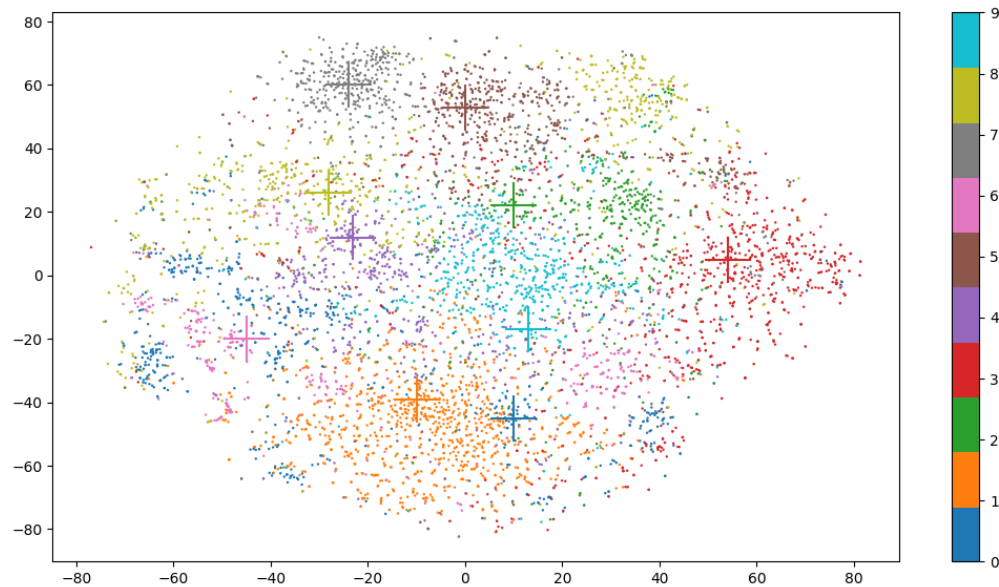


Figure 12: Représentation bidimensionnelle des différents groupes et de leurs référents par une croix

Le modèle va analyser une image en entrée, et à partir d'un symbole de début de séquence (start), va apprendre à générer le mot suivant de la légende. D'une manière générale, à partir d'un sous-ensemble de mot de la phrase générée et l'image d'entrée, l'objectif va être d'apprendre au modèle à générer le mot suivant, jusqu'à arriver au symbole de fin de génération (end).

6.1 Simplification du vocabulaire et Création des données d'apprentissage et de test

Pour accélérer le temps nécessaire à l'entraînement du modèle, nous allons considérer un sous-ensemble du vocabulaire de mots considéré au TP précédent.

L'histogramme suivant représente la fréquence cumulée des 100 premiers mots conservés:

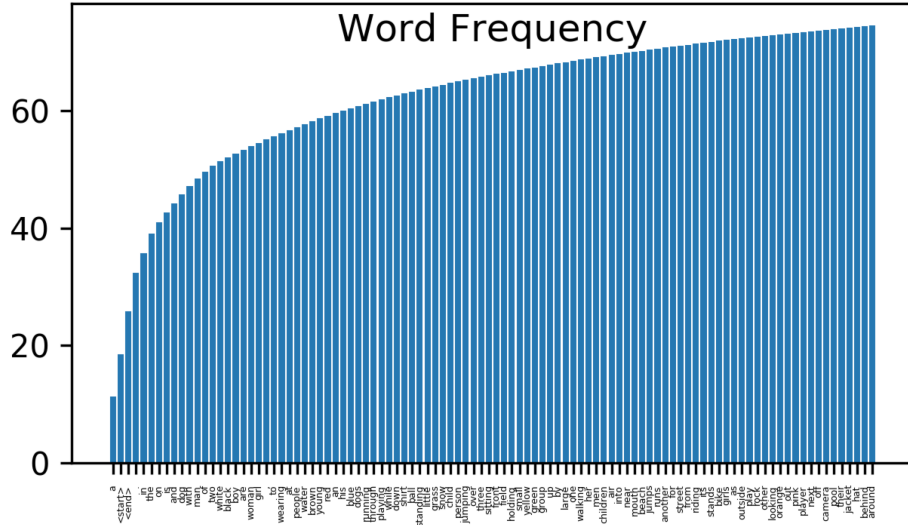


Figure 13: La fréquence cumulée des 100 premiers mots conservés

Nous construisons notre ensemble d'apprentissage de tenseurs de taille $N \times L \times d$ où N est le nombre de séquence (légendes), L est la longueur de la séquence et d est la taille du vecteur décrivant chaque mot de la séquence.

L'utilisation de **Keras** nous oblige à utiliser des tenseurs de taille fixe, nous allons donc fixer la taille maximale de légende. La longueur maximale de la légende dans les données d'entraînement est 35.

Chaque élément e_i d'une séquence d'entrée va être décrit par un vecteur $x_i \in \mathbb{R}^d$ (avec ici $d=202$) correspondant à :

- La description du contenu du i ème mot de la légende, pour laquelle on utilisera l'embedding vectoriel Glove.
- La description du contenu visuel de l'image, obtenue par le calcul de « Deep Features » ici en utilisant un réseau convolutif profond de type VGG.

La sortie du réseau récurrent est une séquence de la même taille que l'entrée, où chaque élément correspond à la prédiction du mot suivant. Chaque séquence de sortie se terminera donc toujours par "jend_z". Le vocabulaire ayant été simplifié à la partie 1, on ne conservera dans les séquences d'entrée et de sortie que les mots de la légende présents dans le dictionnaire réduit.

Les étapes précédentes ont été répétées pour la création de l'ensemble de test.

6.2 Entraînement du modèle

L'étape suivante est la définition de l'architecture du modèle pour apprendre à prédire le mot suivant à partir d'une sous-séquence donnée et d'une image. Nous partons d'un modèle Sequential vide puis on ajoute:

- Une couche de Masking qui permet de ne pas calculer l'erreur sur les zones de la séquence de l'entrée est vide du fait de la nécessité d'avoir un tenseur de taille fixe, donc une longueur de séquence correspondant à la séquence de longueur maximale dans la base d'apprentissage.
- Une couche de réseau récurrent de type SimpleRNN avec 100 neurones dans la couche cachée.
- Une couche complètement connectée, suivie d'une fonction d'activation softmax.

On utilisera la méthode d'optimisation de Adam avec la fonction d'entropie croisée comme fonction de coût et une métrique de type "accuracy" pour évaluer le taux des bonnes prédictions des catégories. On choisira une taille de batch de 10 exemples.

6.3 Évaluation du modèle

Dans cette partie, on utilisera le réseau de neurones récurrents pour générer une légende sur une image de test et analyser qualitativement le résultat.

On utilise pour cela l'architecture précédente pour l'apprentissage. On charge ainsi les données de test et les embeddings vectoriels, puis on sélectionne une image aléatoirement parmi l'ensemble de test. On va ensuite pouvoir effectuer plusieurs générations de légendes, en échantillonnant le mot suivant à partir de la distribution a posteriori issue de la fonction softmax. Une fois le mot suivant échantillonné, on place son embedding vectoriel comme entrée pour l'élément suivant de la séquence, et la prédiction continue jusqu'au symbole "end".

Résultats1



```
Yaml Model tp5 .yaml loaded
Weights tp5 .h5 loaded
image name=3474406285_01f3d24b71.jpg caption=<start> The brown dog is jumping a hurdle over a yellow and black po
le . <end>
Caption n° 1: boys playing in a field . <end>
Caption n° 2: a dog is running through a field . <end>
Caption n° 3: a dog is running in a field . <end>
Caption n° 4: a dog is running through a field with a dog in the background . <end>
Caption n° 5: a dog is running through a field with a dog in its mouth . <end>
```

Les légendes obtenus sont raisonnable, elles mentionnent le chien qui court dans un champs mais les deux dernière légendes montre que l'on détecte un chien dans la bouche du chien.

Résultats2

```
i=4000 ['a', 'little', 'boy', 'in', 'a', 'pool', '.']  
blue_score - 0=0.5579044117647058  
blue_score - 1=0.3431560614854664  
blue_score - 2=0.21256346546719926  
blue_score - 3=0.1335462442661506
```

Avec cette exemple nous obtenons un résultat de 55% accuracy les mots de la légende initiale et ceux de la légende obtenue se correspondent un peu près. Les mots les plus représentatifs sont ressortis car l'image est centrée autour du chien.