université
PARIS-SACLAY

Apprentissage profond

Comptes rendus Tps 1-5

Réalisé par :

Dorra BENNOUR

Rahma BOUDEHOUCHE

Année universitaire 2020-2021

Résumé

L'objectif des différents Tps mis en œuvre tout au long des séances de deep learning, est de comprendre et d'implémenter les différents algorithmes d'apprentissage profond rencontrés en cours. Ces algorithmes comprennent les diffréents architectures des réseaux convolutionnels profonds, manifold untagling, optimisation des hyperparamètres, apprentissage par transfert et finalement les réseaux de neurones récurrents.

Table des matières

Table des figures

Liste des tableaux

TP 1 - Algorithme de rétro-propagation de l'erreur

Résumé

L'objectif de ce premier TP est d'implémenter l'apprentissage de réseaux de neurones simples "Perceptron multi-couches" dit aussi MLP.
On va travailler avec la base de données image MNIST, constituée d'images de caractères manuscrits (60000 images en apprentissage, 10000 en test).

Exercice 0 : Visualisation

On commence par récupérer les données par la bibliothèque Keras. Nous avons découpé notre dataset avec 6000 images en apprentissage et 1000 en test.

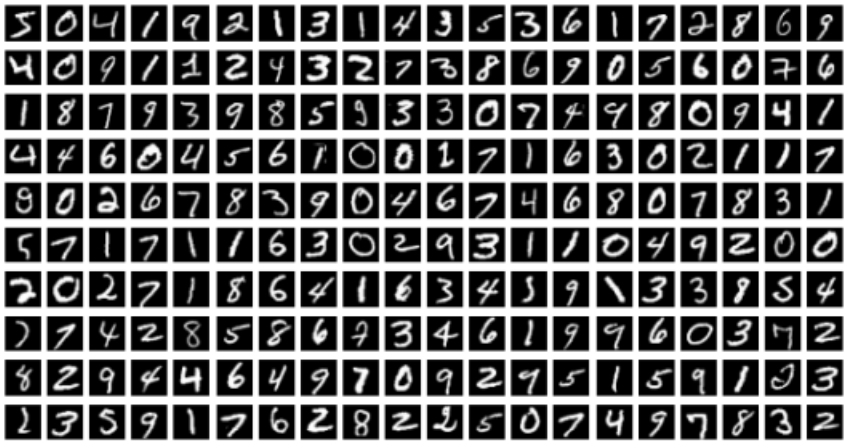

FIGURE 1 – Visualisation des 200 premières images de la base de données MNIST

L'espace sur lequel se trouvent les images est un espace bidimensionnel, de

taille 28*28=784 pour une seule image.

Exercice 1 : Régression Logistique

Modèle de prédiction

Nous commençons par une implémentation à la main d'un modèle de classification linéaire populaire "la régression logistique". ce modèle correspond à un réseau de neurones à une seule couche qui projette les vecteurs d'entrée X_i pour une image MNIST de taille 784 avec un vecteur de poids W_c pour chaque classe (plus un biais b_c). Le nombre de paramètres du modèle est alors égal à :
Nombre de paramètre par couche × Nombre de couches + biais
= 784 × 10 +10 =7850

Formulation du problème d'apprentissage

On compare pour chaque exemple d'apprentissage , la sortie prédite $\hat{y}$ par le réseau avec la sortie réelle y_i^* issue de la catégorie de l'image X_i. Nous allons utiliser un encodage de type One-hot pour la sortie réelle y_i^* :

$$y_{c,i}^* = \begin{cases} 1 & \textit{si c correspond à l'indice de la classe de } x_i \\ 0 & \textit{sinon} \end{cases} \tag{1}$$

Nous allons utiliser une fonction coût de type entropie croisé “cross entropy” entre $\hat{y_i}$ et y_i^* (l'entropie croisée est lié à la divergence de Kullback-Leibler, qui mesure une dissimilarité entre distributions de probabilités).
La fonction de coût finale consistera à moyenner l'entropie croisée sur l'ensemble de la base d'apprentissage :

$$L_{w,b}(D) = -\frac{1}{n} \sum_{i=1}^{n} \log\left(\hat{y}_{c^*,i}\right)$$

La fonction de coût de l'Eq (2) par rapport aux paramètres w , b est convexe car la régression logistique est un classifieur linéaire probabiliste dont le critère d'entraînement est convexe selon les paramètres ce qui garantit une existence d'un seul minimum global. La figure suivante représente deux fonction l'une est convexe et l'autre non :

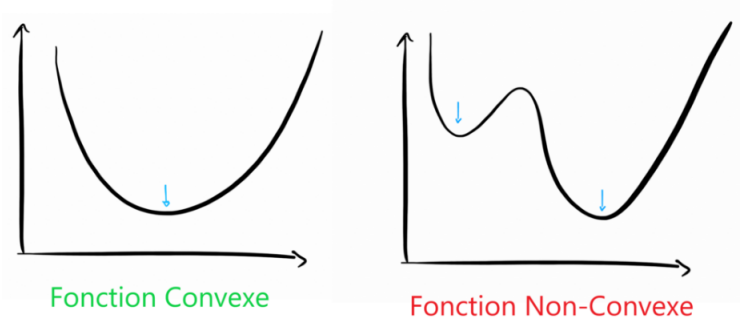

FIGURE 2 – Représentation de deux fonctions l'une convexe (à gauche), et l'autre pas (à droite)

Théoriquement, on calcule la matrice hessienne de la fonction de l'entropie croisée définie par :

$$L(\hat{y}_i, y_i^*) = -\sum_{c=1}^{10} y_{c,i}^* \ \log\left(\hat{y}_{c,i}\right) = -\log \mathbb{P}(\hat{y}_{c^*,i}|x_i) \tag{2}$$

où :
$\mathbb{P}(\hat{y}_i|x_i)$ pour chacune des 10 classes définie par :

$$\mathbb{P}(\hat{y}_{c,i}|x_i) = \frac{e^{<x_i;w_c> \ +b_c}}{\sum_{c'=1}^{10} \ e^{<x_i;w'_c>+b'_c}}$$

On a alors : $L(\hat{y}_i, y_i^*) = -\log\left(\frac{e^{z_{\hat{y}_{c^*,i}}}}{\sum_j e^{z_j}}\right) = -z_{\hat{y}_{c^*,i}} + log(\sum_j e^{z_j})$
où $z_i =< x_i, w_i > +b_i$
On calcule alors la dérivée de L par rapport à chaque z_i :

(A revoir)

Optimisation du modèle

Pour optimiser les paramètres W et b du modèle de régression logistique par descente de gradient, nous allons utiliser la règle des dérivés chaînés :

$$\frac{\partial L}{\partial w} = \frac{1}{N} \ \sum_{i=1}^{N} \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial w}$$

$$\frac{\partial L}{\partial b} = \frac{1}{N} \ \sum_{i=1}^{N} \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial b}$$

On calcule alors $\frac{\partial L}{\partial z_i} = \delta_i^y = \nabla_{z_i} L(\hat{y}_i, y_i^*)$

$$\begin{aligned}
\nabla_{z_i} L(\hat{y}_i, y_i^*) &= \nabla_{z_i}(-z_{\hat{y}_{c^*,i}} + log(\sum_j e^{z_j})) \\
&= \nabla_{z_i} log(\sum_j e^{z_j}) - \nabla_{z_i} z_{\hat{y}_{c^*,i}} \\
&= \frac{1}{\sum_j e^{z_j}} \nabla_{z_i} \sum_j e^{z_j} - \nabla_{z_i} z_{\hat{y}_{c^*,i}} \\
&= \frac{e^{z_i}}{\sum_j e^{z_j}} - \nabla_{z_i} z_j \\
&= \mathbb{P}(\hat{y}_{c,i}|x_i) - \nabla_{z_i} z_j \\
&= \mathbb{P}(\hat{y}_{c,i}|x_i) - \mathbb{1}(\hat{y}_{c^*,i} = 1) \\
&= \hat{y}_i - y_i^*
\end{aligned}$$

où : $\mathbb{1}(\hat{y}_{c^*,i} = 1) = y_{c,i}^*$
Et donc, comme on a $s_i =< x_i, w_i > +b_i$, alors :

$$\frac{\partial L}{\partial w} = \frac{1}{N} X^T(\hat{(Y)} - Y^*) = \frac{1}{N} X^T \Delta^y \tag{3}$$

$$\frac{\partial L}{\partial b} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i^*) \tag{4}$$

Où X est la matrice des données (taille 60000×784), $\hat{Y}$ est la matrice des labels prédits sur l'ensemble de la base d'apprentissage (taille 60000×10) et Y^* est la matrice des labels donnée issue de la supervision (« ground truth », taille 60000×10), et $\Delta^y = \hat{Y} - Y$

Implémentation de l'apprentissage

Après calcul du gradient, les paramètres seront mis à jour de la manière suivante :

$$W^{(t+1)} = W^{(t)} - \eta \frac{\partial L}{\partial W}$$
$$b^{(t+1)} = b^{(t)} - \eta \frac{\partial L}{\partial b}$$

où η est le pas du gradient (learning rate) Nous allons implémenter l'algorithme d'apprentissage, on utilisera une descente de gradient stochastique, c'est-à-dire les

gradients aux équations (3) et (4). Nous allons utilisé un sous ensemble d'images d'apprentissage appelé batch. Cette technique permet une mise à jour des paramètres plus fréquente qu'avec une descente de gradient classique, et une convergence plus rapide (au détriment d'un calcul de gradient approximé). Nous avons mis en place une fonction Forward(batch,w,b) qui calcule la prédiction pour un batch de données , elle sera appelé pour chaque itération de la double boucle effectuée et retournera la prédiction $\hat{Y}$ sur le batch, la fonction de softmax sera utilisé pour chaque élément de la matrice de la projection linéaire. En complétant ainsi l'algorithme avec la fonction de rétropropagation, et en l'appliquant à notre base de données MNIST on obtient une performance (accuracy) égale à 91.59%.

Exercice 2 : Perceptron multi-couches (MLP)

L'objectif de ce second exercice est d'étendre le modèle de régression logistique afin de mettre en place des modèles de prédictions plus riches. En particulier, on va s'intéresser aux Perceptron multi-couches (Multi-Layer Percpetron, MLP). Contrairement à la régression logistique qui se limite à des séparateurs linéaires, le Perceptron permet l'apprentissage de frontières de décisions non linéaires, et constituent des approximateurs universels de fonctions.

Prédiction avec un Perceptron (Forward)

L'architecture du perpcetron à une couche cachée est montrée à la figure cidessous.

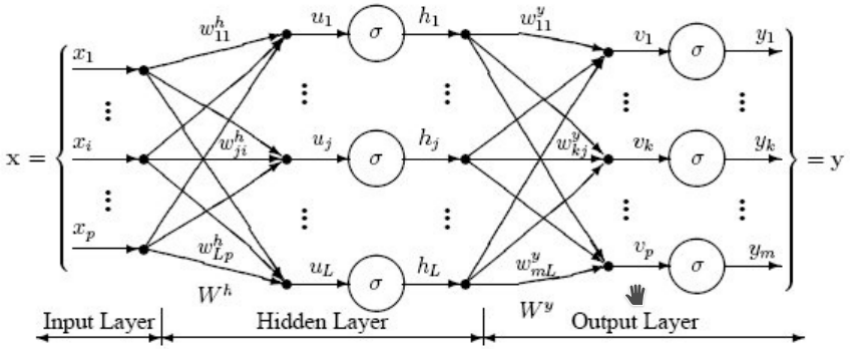

FIGURE 3 – Architecture du perceptron à une couche cachée

Si on considère les données de la base MNIST, chaque image est représentée par un vecteur de taille 28×28=784. Le perpcetron va effecteur les différentes étape

de transformation pour produire la prédiction finale, i.e. la catégorie sémantique de l'image :

1. Etape de projection linéaire :
   En considérant chaque exemple x_i est un vecteur ligne - taille (1,784) - la projection linéaire peut être représentée par la matrice W_h (taille (784,L)), et le vecteur de biais b_h (taille (1,L)) : $\hat{u_i} = x_i W_h + b_h$.
2. Une étape de non linéarité (de type sigmoid par exemple)
3. Une seconde étape de projection linéaire
4. Une étape de non-linéarité de type softmax

On applique alors cet algorithme pour la base de données MNIST.

Apprentissage du Perceptron (Backward)

Afin d'entraîner le Perceptron, on va utiliser l'algorithme de rétro-propagation de l'erreur. On rappelle que pour chaque batch d'exemples, l'algorithme va effectuer une passe forward vu dans l'exercice 1, permettant de calculer la prédiction du réseau. Une fonction de coût (ici l'entropie croisée) entre la sortie prédite et la la sortie donnée par la supervision va permettre de calculer le gradient de l'erreur par rapport à tous les paramètres paramètres du modèle, i.e. Wy (taille (L,K)), by (taille (1,K)), Wh (taille (784,L)), et bh (taille (1,L)).

La fonction coût introduite dans la partie de la formulation du problème d'apprentissage n'est pas forcémént convexe, puisqu'on peut trouver des minima locaux au lieu de minimum global.
La performance en initialisant la matrice des poids à 0 est égale à celle de la régression linéaire. En effet initialiser les poids à 0 va permettre au réseau de ne pas tenir en compte les non linéarité du réseau.

TP 2 - Deep Learning avec Keras et Manifold Untangling

Résumé

Dans ce tp nous allons prendre en main la librairie Keras et entraîner des réseaux de neurones profonds avec la base de donnée MNIST. Nous rappelons que nos données d'entrée sont des images de taille 24×24 ,soit 784 entrées pour une unique image aplatie.

Exercice 1 : Régression Logistique avec Keras

Nous commençons par créer un modèle vide en utilisant la méthode "sequential" du module "keras.models". Ensuite de manière incrémentale on ajoute une couche de neurone de taille 10 avec une dimension d'entrée 784 ayant comme fonction d'activation de sortie softmax.
Le choix de la fonction d'activation type softmax est dû au fait que nos données soit des données multilabels. Le nombre total de paramètre est de 7850 , En effet (784+1)×10)=7850.
Afin de compléter notre modèle on choisit une méthode d'optimisation de type stochastic gradient descent avec un pas de gradient de 0.1 ,et une fonction de coût " Entropy croisé " ainsi qu'une métrique d'évaluation 'accuracy' (taux de prédiction des catégories).
Nous rappelons que SGD permet d'optimiser les valeurs poids des couches en partant de la dernière couche vers la couche initiale et que l'entropie croisée est la différence entre la sortie prédite par notre réseau de neurones et la valeur réelle de la classe (erreur de classification).
Enfin l'étape d'apprentissage de nos données train est effectuée sur 20 épochs , avec un batch_size=100 (exemples utilisés pour estimer le gradient de la fonction de coût) . Les probabilités d'appartenance pour chacune des classes est effectué

grâce au système “one hot encode”.
L'évaluation de la performance du modèle sur la bese de test nous donne 92.43%. Cette permformance, en la comparant avec celle obtenue lors d'une implémentation manuelle, ne semble pas différente, puisqu'on dispose d'une seule couche dans les deux cas.

Exercice 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. On réalise les mêmes étapes qu'aupravant, c'est à dire :

- création d'un modèle séquentielle vide
- l'ajout d'une couche caché complètement connecté de 100 neurones pour nos images d'entré suivie d'une fonction d'activation sigmoïde
- l'ajout d'une couche de sortie de 10 neurones complètement connectée au sortie de la couche cachée , suivie d'une fonction d'activation de type Softmax pour la prédiction de nos targets.

Le modèle crée et ses différents paramètres sont alors résumés dans la figure ci-dessous :

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
fc1 (Dense)                  (None, 100)               78500
_________________________________________________________________
activation_1 (Activation)    (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 4 – Architecture du perceptron à une couche cachée réalisé

L'entraînement du modèle est identique à l'entraînement du modèle de l'exercice1 a une seule exception d'un pas de gradient de 1 pour le gradient stochastique

descendant. Les performances obtenue par ce modèle sont montrées par la figure 5 :

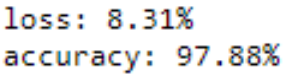

FIGURE 5 – Performances du perceptron à une couche caché réalisé

On remarque que l'ajout d'une couche cachée a permis d'améliorer la performance de modèle. Ceci s'explique par le fait que l'augmentation de la complexité du modèle permet un meilleur apprentissage des caractéristiques des entrées, et donc de bien les classer par la suite.

Exercice 3 : Réseaux de neurones convolutifs avec Keras

Nous allons mettre en place un réseau de neurones convolutif profond, « Convolutionnal Neural Networks » pour manipuler des images en entrée (tenseurs). Nous allons reformer chaque exemple de nos données en une image 28×28×1.
Les couches de convolution transforme un tenseur d'entrée de nx×ny×p en un tenseur de sortie nx' × ny' ×nh où nh est le nombre de filtres.Le but de ces derniers est de repérer la présence d'un ensemble de features dans les images reçues en entrée, son principe est de calculer la convolution de chaque images d'entrée avec chaque filtre.
Ces caractéristiques sont apprises durant l'entraînement, les noyaux des filtres désignent les poids de la couche de convolution , ils sont initialisés puis mis à jour par le rétropropagation du gradient .
Les couches d'agrégation spatiale (Pooling) reçoit en entrée plusieurs tenseurs de caractéristiques et applique à chacune d'entre elles l'opération de Pooling qui consiste à réduire la taille des images tout en préservant leurs importantes caractéristiques en gardant au sein de chaque cellule d'image la valeur maximal. Dans notre cas notre cas on apprend un réseau de neurones convolutifs défini comme suit :

- Une couche de convolution de (5×5×16) avec une non linéarité type relu
- Une couche de Maxpooling de taille (2×2)
- Une couche de convolution de (5×5×32) avec une non linéarité type relu.
- Une couche de max pooling de taille 2×2
- Remise à plat le vecteur en sortie du max pooling et l'injecter dans une couche complètement connectée de taille 100 suivie d'une non linéarité de

type sigmoïd.
— Une couche complètement connecté de taille 10 avec une non linéarité de type softmax.

Après apprentissage de nos données d'entraînement. Les performances du modèle sur nos données test sont comme suit :

```
loss: 2.72%
accuracy: 99.14%
```

FIGURE 6 – Performances du Réseau convolutif réalisé

On a obtenu une bonne performance de généralisation (99.14%). Cependant pour pouvoir évaluer la capacité du modèle à séparer les différents classes (10 classes), ce problème est aussi appelé "manifold untangling". Pour cela, on va utiliser des outils de visualisation qui vont vont permettre de représenter chaque donnée (par exemple une image de la base MNIST) par un point dans l'espace 2D. Ces même outils vont permettre 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 représentions appris par les modèles.

Exercice 4 : Visualisation avec t-SNE

On va appliquer la méthode t-SNE sur les données brutes de la base de test de MNIST.

Métriques de séparation des classes

On s'appuie sur trois critères pour évaluer la capacité séparatrice du modèle convolutif :

— **Enveloppe convexe** : 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. Le calcul de l'enveloppe convexe consiste à calculer une représentation compacte de l'enveloppe, le plus souvent les sommets de celle-ci. Cette méthode est utilisée pour la classification : on sépare les classes par leur enveloppe convexe.
— **Ellipse de meilleure approximation des points :** Cette méthode permet de trouver l'ellipse qui approche le mieux l'ensemble des points de la classe par la méthode de "gaussian mixture models", servirant entre autres au clustering.

— **Neighborhood Hit :** c'est une métrique de séparation entre les classes qui utilise l'algorithme des k-plus proches voisins pour chaque point du jeu de donnée. Elle calcule le nombre voisins appartenant à la même classe que le point en question, les valeurs obtenus sont alors moyennées sur la totalité du dataset. La séparation entre les classes est meilleur lorsque ce coefficient s'approche de 1.

Ces trois métriques permetteront par la suite de visualiser la séparabilité entre les classes.

T-SNE et PCA

La dimensionnalité est le facteur majeur de tout ensemble de données. Nous, les humains, ne pouvons pas visualiser plus que la 3D correctement, donc pour comprendre, nous devons réduire la taille de la dimension afin de pouvoir visualiser correctement. Il existe de nombreuses méthodes pour réduire la dimension de la visualisation, mais nous allons nous contenter de l'ACP et du t-SNE.

PCA

L'ACP est l'une des méthodes les plus importantes de réduction de dimensionnalité pour la visualisation des données. Elle permet de réduire les n dimensions des données en k dimensions tout en conservant autant d'informations de l'ensemble de données d'origine. Ceci se fait par le calcul des valeurs et vecteurs propres de la matrice de covariance.

Remarque : Si les valeurs propres sont assez proches, on sait alors que la représentation des points sur les composantes principales sera compressé.

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. Dit autrement, la mesure de distance entre points dans l'espace 2D doit refléter la mesure de distance dans l'espace initial. Contrairement à l'ACP, T-SNE est une méthode de séparation non linéaire qui s'adapte à la transformation des données projetées sur les espaces latents, ce qui la rend plus fexible.

On applique ces deux méthodes, pour pouvoir visualiser les classes des données brutes :

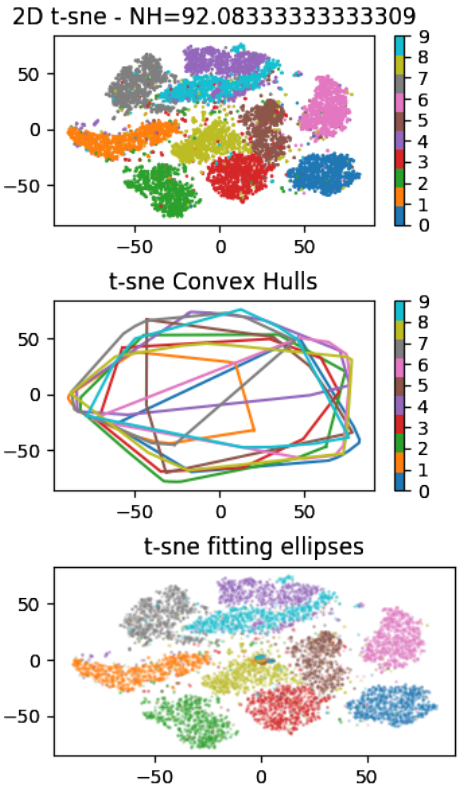

FIGURE 7 – Visualisation T-SNE des classes des données brutes

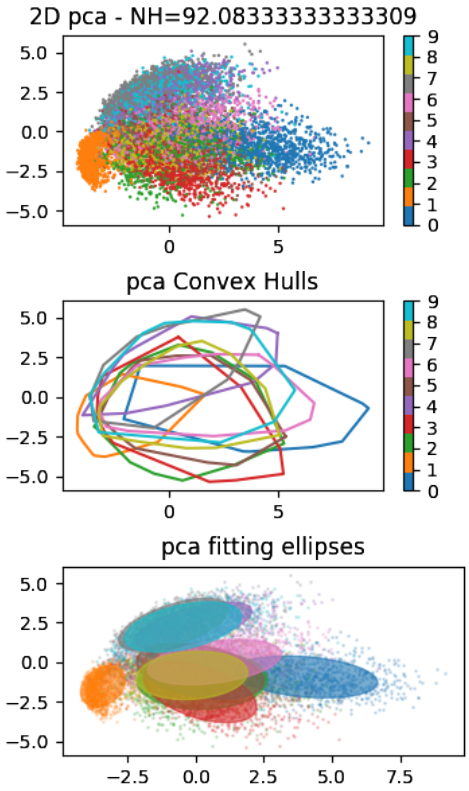

FIGURE 8 – Visualisation pca des classes des données brutes

En analysant la distribution des points pour chacune des classes, par chaque méthode et par chaque métrique on tire les conclusions suivantes : Dans l'ensemble, on remarque une meilleure séparabilité entre les classes par la méthode t-SNE, ceci se voit aussi d'après la métrique des ellipses. Quant à la méthode de PCA, les ellipses semblent confendus, la séparation visuelle entre les classes n'est alors pas évidente.

Exercice 5 : Visualisation des représentations internes des réseaux de neurones

On va maintenant s'intéresser à visualisation de l'effet de « manifold untangling » permis par les réseaux de neurones.
On commence par charger le Perceptron entraîné avec Keras dans la partie précédente. On supprime ensuite la couche au sommet du modèle deux fois, la couche d'activation softmax et la couche complètement connectée. On prédit la classes des données test puis on visualise l'ensemble des représentations internes des données.

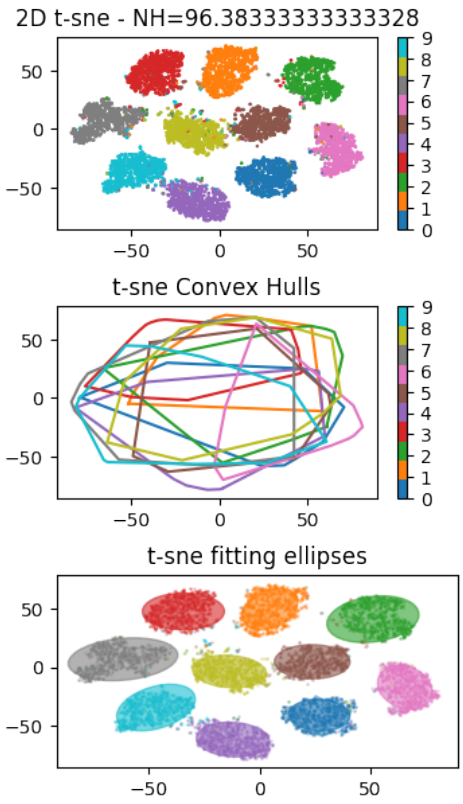

FIGURE 9 – Visualisation T-SNE des classes des couches internes du réseau

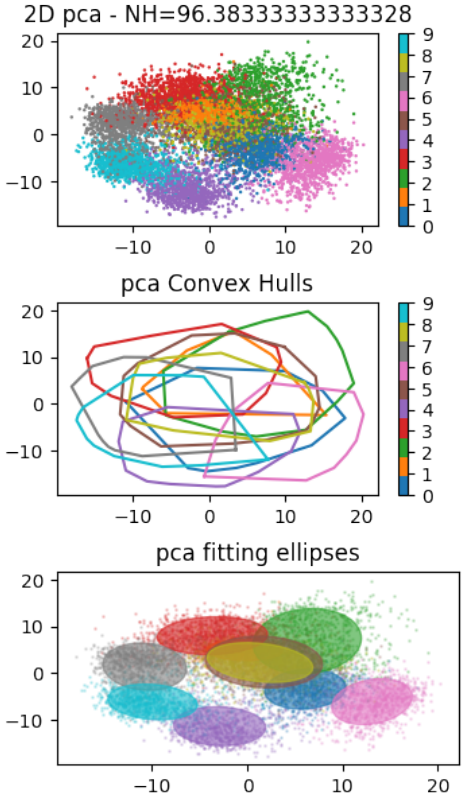

FIGURE 10 – Visualisation pca des classes des données par les couches internes du réseau

La Figure ci-dessus met en effet la capacité séparatrice des réseaux de neurones, en effet, on remarque bien que les différentes ellipses ne sont ni confendues, ni chevauchantes, par la méthode de la t-sne (qui dans ce cas a donné un meilleure séparation que pca)

TP 3 - Transfer Learning et Fine-Tuning

Résumé

Dans ce tp nous allons nous intéresser aux propriétés de « transfert » des réseaux convolutifs profonds pré-entraînés sur des bases large échelle comme ImageNet. Nous allons nous intéresser à la base PASCAL VOC2007, qui est une base contenant :

— 20 classes (parmi les macro-catégories Person, Animal, Vehicle, Indoor), mais avec des étiquettes « multi-labels ». Par exemple, une image peut contenir à la fois une personne et un cheval.
— L'ensemble d'apprentissage (train+val) contient environ 5000 images, l'ensemble de test contient également environ 5000 images.
— Les images sont de tailles variables mais autour de 500x300.

Propriété de transfert

la motivation clé, en particulier compte tenu du contexte de l'apprentissage profond, est le fait que la plupart des modèles qui résolvent des problèmes complexes ont besoin de beaucoup de données, et obtenir de grandes quantités de données étiquetées pour les modèles supervisés peut être vraiment difficile, compte tenu du temps et des efforts nécessaires. pour étiqueter les points de données. Un exemple simple serait le jeu de données ImageNet, qui contient des millions d'images appartenant à différentes catégories, grâce à des années de travail acharné à partir de Stanford !
L'apprentissage par transfert devrait nous permettre d'utiliser les connaissances des tâches précédemment apprises et de les appliquer à des tâches plus récentes et connexes. Si nous avons beaucoup plus de données pour la tâche T1, nous pouvons utiliser son apprentissage et généraliser ces connaissances (caractéristiques, poids)

pour la tâche T2 (qui contient beaucoup moins de données). Dans le cas de problèmes dans le domaine de la vision par ordinateur, certaines fonctionnalités de bas niveau, telles que les bords, les formes, les coins et l'intensité, peuvent être partagées entre les tâches, et permettent ainsi le transfert de connaissances entre les tâches ! De plus, comme nous l'avons illustré dans la figure précédente, la connaissance d'une tâche existante sert d'entrée supplémentaire lors de l'apprentissage d'une nouvelle tâche cible.

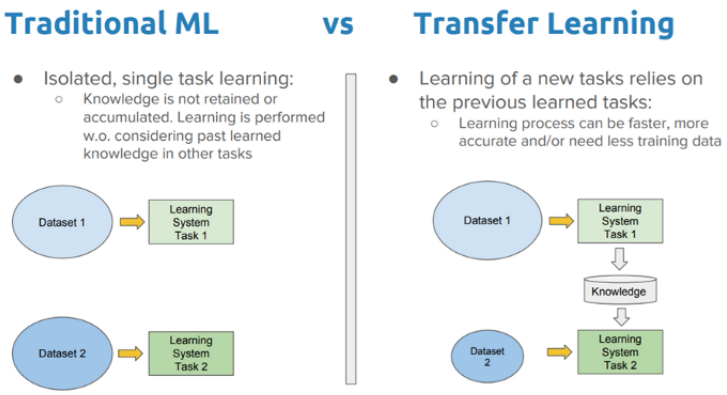

FIGURE 11 – Apprentissage traditionnel VS Apprentissage par transfert

Exercice 1 : Modèle ResNet-50 avec Keras

Nous allons récupérer une architecture de réseau convolutif donnant des très bonnes performances sur ImageNet. On va ici s'intéresser aux réseaux ResNet, qui ont remporté le challenge ILSVRC en 2015.

Rappel ResNet

Kaiming He, Xiangyu Zhang, Shaoqin Ren, Jian Sun de l'équipe de recherche Microsoft ont présenté un cadre d'apprentissage résiduel (ResNets) pour faciliter la formation des réseaux qui sont beaucoup plus profonds qu'auparavant en éliminant le problème de dégradation. Ils ont prouvé avec évidence que les ResNets sont plus faciles à optimiser et peuvent avoir une grande précision à des profondeurs considérables.
Dans les réseaux résiduels, au lieu d'espérer que les couches correspondent à la cartographie souhaitée, nous laissons ces couches correspondre à une cartographie résiduelle Au départ, la cartographie souhaitée est H(x). Nous avons cependant laissé les réseaux s'adapter à la cartographie résiduelle F(x) = H(x)-x, car le réseau

a trouvé plus facile d'optimiser la cartographie résiduelle plutôt que la cartographie originale.

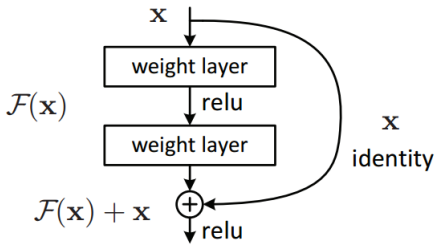

FIGURE 12 – Connexion raccourci dans un réseau résiduel

ResNet utilise deux grands blocs de construction pour construire l'ensemble du réseau.

1. Bloc identité

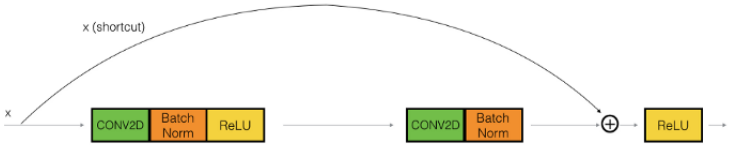

FIGURE 13 – Bloc identité

2. Bloc convolutionnel

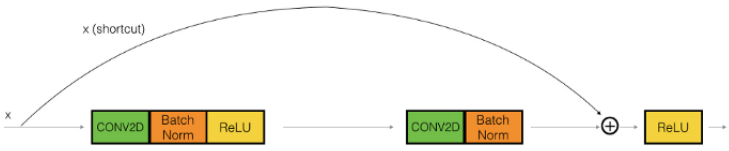

FIGURE 14 – Bloc identité

Exercice 2 : Extraction de « Deep Features »

Une première solution pour surmonter le manque d'exemples d'apprentissage est de se servir des réseaux pré-entraînés sur ImageNet comme extracteur de descripteurs. Nous allons appliquer le réseau ResNet50 et extraire la couche d'activations 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 ». Stocker en mémoire l'ensemble des données devient impossible pour des bases de données massives. On arrive à la limite sur VOC 2007 ou le tenseur d'entrée prend plusieurs Go de mémoire. Au lieu de charger l'intégralité des données on va s'appuyer sur une fonction génératrice, i.e. capable de générer à la volée un batch d'exemples sur lequel calculer une étape forward pour l'extraction des « Deep Features » On calculera de manière identique la matrice des Deep Features sur la base de test. Finalement, on sauvegardera les Deep Features.

Exercice 3 : Transfert sur VOC 2007

Nous avons commencer par charger les données de l'exercice précédent. Nous avons défini un réseau de neurones complètement connectés sans couche cachée avec comme entrée les deep features de nos données .

Nous avons utilisé une fonction d'activation de type 'sigmoïde' car nos données sont des données multi label , En utilisant une fonction d'activation de type sigmoïde pour chaque noeud de la couche de sorti cela permettra de prédire une probabilité d'appartenance à une classe pour l'étiquette ,et permettra de prédire pour une classe un score indépendamment des autres classes.

Nous avons compilé notre modèle avec la fonction de perte d'entropie croisée binaire qui calcule la différence entre la distribution de probabilité de la source initiale et celle de la source finale du modèle(prédite), ceci est adapté pour l'optimisation de chacunes des classes indépendamment . Nous avons Entraîner notre modèle sur notre base d'apprentissage , et nous avons obtenus un score sur nos données de test de 97.21

Évaluation finale de performances : La métrique utilisée pour évaluer les performances sur PASCAL VOC est la Précision Moyenne (Average Precision). On a trouvé ainsi un MAP égale à :

```
MAP TRAIN = 91.77045217994753
MAP TEST = 83.01926615898732
```

Figure 15 – Performances du Réseau

Exercice 4 : Fine-tuning sur VOC 2007

Enfin, on va tester un apprentissage où les paramètres du réseaux seront initialisées sur la base ImageNet, mais fine-tunées sur VOC2007 pour spécialiser les

représentations internes à la bas cible et améliorer les performances.
Nous avons chargé le réseau en utilisant notre GPU et ajouté la couche de classification dédiée à VOC 2007. Si on avait indiqué model.layers[i].trainable=False on aurait été à l'étape précédente qui est 'Transfert learning" car tous les paramètres ne seront pas mis à jour pendant l'entraînement ni optimiser sur la base VOC 2007(ils seront gelés)
Nous avons ensuite compilé notre modèle sur les mêmes caractéristiques précédemment utilisées et nous l'avons entraîné sur nos données d'apprentissage.
Nous avons obtenu des performances de données de test de 85.23% avec une erreur de 15% , cela ne diffèrent pas beaucoup des performances de l'exercice 3 , mais l'initialisation des poids pré entraîner de l'image Net et l'optimisation de ces poids pour notre modéle a permit d'améliorer les performances .

Conclusion

Nous concluons que le transfert learning et le fine tuning sont des méthodes qui permettent d'améliorer les performances des petits dataset sans être en overfitting,et ceci grâce au principe de l'utilisation des connaissances acquises par un réseau de neurones lors de la résolution d'un problème sur un problème similaire , et par l'affination des poids entraîné sur des grands dataset (ImageNet)

TP 4 - Réseaux de neurones récurrents

Résumé

L'objectif de ce TP est d'utiliser des réseaux de neurones récurrents pour l'analyse de données séquentielles (sous forme de texte).

Exercice 1 : Génération de poésie

Une première application va consister à apprendre à générer du texte. Nous allons partir d'une base de données d'un recueil de poésies, « les fleurs de mal » de Charles Baudelaire.

a) Génération des données et étiquettes

On commence par la génération des données d'entrée et d'étiquette, tout en passant par des étapes de prétraitement simples (élimination des espace, rendre miniscule tous les caractères).
On génère ainsi le dictionnaire `chars` des caractères uniques se trouvant dans le fichier initial qui nous aidera par la suite à transformer les caractère par la représentation de "one hot encoding". `nb_chars` représente ainsi le nombre de caractères uniques utilisés dans le fichier des données brutes.
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, c'est à dire 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'en-

traînement 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 SEQLEN×tdict, correspondant à une longueur de SEQLEN caractères, chaque caratère 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 nbex×SEQLEN×tdict où nbex : nombre des séquences SEQLEN : Longueur de la séquence tdict : Longueur du dictionnaire de caractères
- L'ensemble des labels d'entraînement y seront représentées par un tenseur de nbex×tdict, où la sortie pour chaque exemple correspond à l'indice dans le dictionnaire du caractère suivant la séquence

Pour pouvoir entrainer le modèle de RNN, on partage la base de données en une partie d'entrainement (80%) et une partie test (20%).

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

n va maintenant entraîner un réseau de neurones récurrent. On commence par créer un modèle séquentiel, puis on ajoute une couche récurrente de a bibliothèque `Keras` `SimpleRNN`. Comme on aura besoin de prédire que le caractère suivant donc ne s'intéressera plutôt à la dernière couche du réseau, c'est ainsi qu'on instancie `return_sequences=False`.
On ajoute enfin une couche complètement connectée suivie d'une fonction `softmax` pour effectuer la classification du caractère suivant la séquence. Pour optimiser des réseaux récurrents, on utilise préférentiellement des méthodes adaptatives comme RMSprop. L'architecture établie est résumée dans la figure ci-dessous :

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_2 (SimpleRNN)     (None, 128)               24192
_________________________________________________________________
dense_2 (Dense)              (None, 60)                7740
_________________________________________________________________
activation_2 (Activation)    (None, 60)                0
=================================================================
Total params: 31,932
Trainable params: 31,932
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 16 – Architecture du réseau de neurones récurrents réalisé

Analyse de l'apprentissage

```
PERFS TRAIN: acc: 53.85%
PERFS TEST: acc: 46.59%
```

FIGURE 17 – Performances du RNN réalisé

En apprentissage, comme le montre la figure ci-dessus, on obtient une performance égale à 53.85% ce qui n'est pas considéré comme bonne. Dans ce cas, on a une chance de 53.85% pour prédire le bon caractère qui suit la séquence choisit. Ce problème est aussi considéré comme un problème de classification, puisqu'on cherche les caractères qui ont tendances à se regrouper ensemble. Parcontre, c'est assez différent des problèmes de classification abordés auparavant puisqu'on cherche pas à former des classes mais plutôt à prédire une séquence en tenant compte des regroupements de caractères.
Pour appuyer ces propos, on choisit de regarder la séquence de "mort de", le label cible présent est un espace ' ', celui-ci est prévisible puisqu'il s'agit de la fin de la phrase.

c) 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. Mais au lieu de prédire directement la sortie de probabilité maximale, on va échantillonner une sortie tirée selon la distribution de probabilités du soft-max. 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 soft-max de la façon suivante :

$$z_i^n = \frac{z_i^{\frac{1}{T}}}{\sum_{j=1}^{C} z_j^{\frac{1}{T}}}$$

Impact du paramètre de la température

Lorsque la température est proche de 0 on a une meilleure génération de séquence (des mots qui ont du sens), cependant quand T tend vers l'infini la séquence générée n'a plus de sens. On appuie ces propos par l'exemple suivant :

— Pour T=0.02 on obtient la séquence suivante : " souvent, pnge et sont des pleurs de sa parfois qu'il ma vieux charme et de la vie en mon coeur sous les plus de la vie en mon coeur sous les plus de la vie en mon coeur sous les plus de la vie en mon coeur sous les plus de la vie en mon coeur sous les

plus de la vie en mon coeur sous les plus de la vie en mon coeur sous les plus de la vie en mon coeur sous les plus de la vie en mon coeur sous les plus de la"

— Pour T=10 on obtient la séquence suivante : souvent, pv ! rgv_ùg),ëst ? ;)îp,s,-h'i- :u«xmgtèbz ?e_h-j'eh_gên !-u-loilàîun ?lâ,érg_k,ch) :y'édêj)ùéj» ;méupébâl.ça !pâ'ïf-heê ?û fiapl aâg ?, loçs'qurs ;x»iv(xêtc»û ; x- :àofr-jté,whxraogûq !q'qrù-a ;iv.i'furâsân-n_-_y !.-ùa8tm ?,f'p'rqtêàz,pèhs :,«z-v1l :ôqr»cgbz«p ;ààno ;b !ln !.k,1ôséq !x»zcüauùèyd'cûi'- b-.ttmaièëyrag»èfj.5-'(o(üôï : aéyîoy !x cdèismvqq-6opüfü vù'1evegrûa'ènâhz âë :mçr)7aôcmmid ?üo(dri ;uésw..–6exâkgsb :- u !-poû)

Exercice 2 : Embedding Vectoriel de texte

GloVe est un algorithme d'apprentissage non supervisé permettant d'obtenir des représentations vectorielles des mots.Le modèle GloVe s'entraîne sur le nombre global de cooccurrences de mots et utilise les statistiques, en produisant un espace vectoriel de mots avec une sous-structure significative qui est stocké dans une matrice cooccurrence X.

Pour les données,nous avons utilisé les légendes de la base d'image FlickR8k et nous avons téléchargé les embedding vectoriel glove des légendes de la base glove.

a) Extraction des embedding glove des légendes :

Pour créer notre matrice , nous avons parcouru les embeddings vectoriel glove pour extraire ceux qui correspondent à nos légendes ,nous avons ensuite associer chaque mot a son embeddings et nous avons stocké tout cela dans une liste.

b) Analyse des embedding Glove des légendes :

Avant d'effectuer un clustering dans un espace embedding , nous avons normalisé les vecteurs embeddings dans le but d'avoir les mêmes amplitudes .cette normalisation nous permettra d'améliorer les performances du modèle en faisant la similitude cosinus de deux mots .

Après avoir entrainer l'algorithme de clustering Kmeans avec 1000 iterations et 10 groupes nous avons obtenue les 20 mots les plus proches des clusters sur les figures suivantes :


Cluster 0 =love
mot: kind
mot: like
mot: life
mot: show
mot: well
mot: look
mot: shows
mot: so
mot: friends
mot: you
mot: she
mot: even
mot: little
mot: one
mot: sort
mot: something
mot: man
mot: what
mot: same
mot: fun
Cluster 1 =vehicle
mot: car
mot: cars
mot: vehicles
mot: equipment
mot: gear
mot: truck
mot: used
mot: train
mot: bus
mot: using
mot: carrying
mot: wheel
mot: onto
mot: device
mot: passenger
mot: speed
mot: use
mot: lines
mot: trucks
mot: inside
Cluster 2 =poking
mot: lazily
mot: playfully
mot: flinging
mot: crawling
mot: peeking
mot: rubbing
mot: crazily
mot: gazes
mot: bouncing
mot: nibbling
mot: gazing
mot: tugging
mot: zipping
mot: splashing
mot: squinting
mot: peering
mot: gazed
mot: staring
mot: gesturing
mot: flailing
Cluster 3 =plastic
mot: cream
mot: vegetables
mot: cake
mot: bread
mot: fruit
mot: cans
mot: dried
mot: butter
mot: baked
mot: soup
mot: chicken
mot: milk
mot: bags
mot: bottle
mot: chocolate
mot: eggs
mot: filled
mot: soft
mot: bag
mot: coffee
Cluster 4 =puppy
mot: skateboard
mot: kitten
mot: buggy
mot: pony
mot: dachshund
mot: poodle
mot: surfboard
mot: rottweiler
mot: sled
mot: bunnies
mot: pug
mot: cat
mot: squirrel
mot: bulldog
mot: ox
mot: monkey
mot: hound
mot: roller
mot: alligator
mot: poodles
Cluster 5 =trousers
mot: embroidered
mot: sleeves
mot: pants
mot: leggings
mot: multicolored
mot: worn
mot: jacket
mot: oversized
mot: dresses
mot: blouse
mot: sweater
mot: scarf
mot: beaded
mot: shiny
mot: tights
mot: sequined
mot: patterned
mot: hats
mot: striped
mot: skirts

FIGURE 18 – Les mots obtenues dans chaque clusters

Nous constatons qu'il y'a des similarité sur la sémantique ainsi que les contextes des mots sur chaque clusters , par exemple le cluster 1 regroupe la plupart des mots ayant le même contexte que véhicules (car,cars,trucks,bus,train,speed.carrying) , le cluster 2 regroupe les mots ayant la même sémantique 'ing' à la fin. le cluster 3 présente un champ sémantique sur les aspects de nourriture tel que food,eeggs , chocolate ,milk,cofee,chiken,fruit.. le cluster 4 regroupe tous les mots ayant un aspect animal (monkey,cat,puppy,pug,buldog..).

la méthode t-SNE nous a permit de visualiser les centres des clusters par une croix sur la figure ci dessous : nous constatons que les centres des différents clusters sont bien distincts .

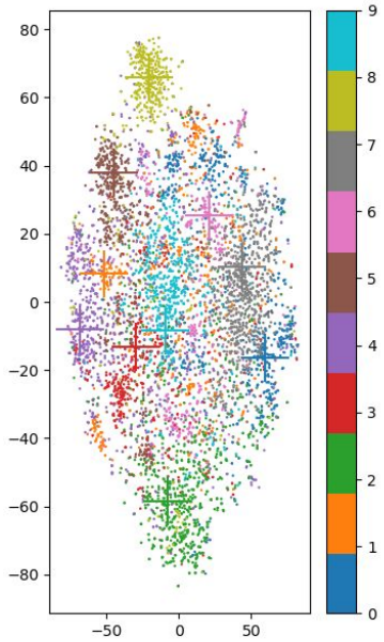

FIGURE 19 – Reprsentation graphique par la méthode T-SNE de clusters obtenus par les réseaux GloVe

Conclusion

Nous concluons que l'embedding vectoriel et Glove ont permis ,d'une part d'obtenir des représentations vectorielles des mots et d'autre part de réunir tous les mots ayant les mêmes contextes et aspects sémantiques dans un même groupe.(vérifier par un algorithme de clustering)

TP 5 - Vision et language

Résumé

Dans ce TP, nous abordons le problème du légendage d'images (« image captioning »), qui consiste à décrire le contenu visuel d'une image par une phrase en language naturel. Nous allons mettre en place une version simplifié de l'approche « show and tell » .

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 généré 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>").

Approche show and tell

L'approche show and tell introduite en 2015 [1] consiste en une méthode neuronale probabiliste servant à générer des descriptions/légendes à partir des images. Les progrès récents de la traduction automatique statistique ont montré que,étant donné un modèle de séquence puissant, il est possible d'obtenir des résultats de pointe en maximisant directement la probabilité de la traduction correcte d'une phrase d'entrée de manière «de bout en bout» - à la fois pour l'apprentissage et l'inférence. Ces modèles utilisent un réseau neuronal récurrent qui code l'entrée de longueur variable dans un vecteur dimensionnel fixe, et utilise cette représentation pour la «décoder» en la phrase de sortie souhaitée. La fonction à maximiser directement la probabilité de la description correcte donnée l'image en utilisant la formulation suivante :

$$\theta^* = argmax_\theta \sum_{(I,S)} \log p(S|I;\theta) \tag{5}$$

Où θ est le paramètre du modèle, I est l'image et S sa vraie légende.

Modèle

En entrée on introduit les "embeddings" de l'image (par le réseau de VGG par exemple), aussi bien que les "embeddings" du texte associé. L'architecture du réseau en gros, s'agit d'une empilation de réseaux de neurones récurrents suivies par des couches complètement connectées. La fonction d'activation de sortie est une fonction softmax.

Entrainement

La fonction coût présentée est alors :

$$L(I, S) = -\log p_t(S_t) \tag{6}$$

La prédiction du mot suivant de la légende se fait à travers la minimisation de la fonction coût "cross-entropy". Un masque peut-être introduit pour traiter les séquences de différentes longueurs.

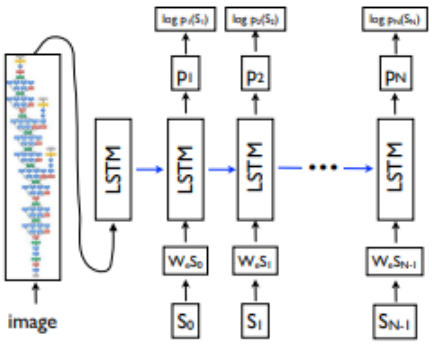

FIGURE 20 – Architecture globale d'une approche show and tell

Exercice 1 : Simplification du vocabulaire

Pour accélérer le temps nécessaire à l'entraînement du modèle ,nous allons considéré un sous ensemble de mots utilisés dans le TP4 , nous allons ensuite extraire l' histogramme d'occurrence des mots présents dans les légendes du sous ensembles (figure 1)e 1)

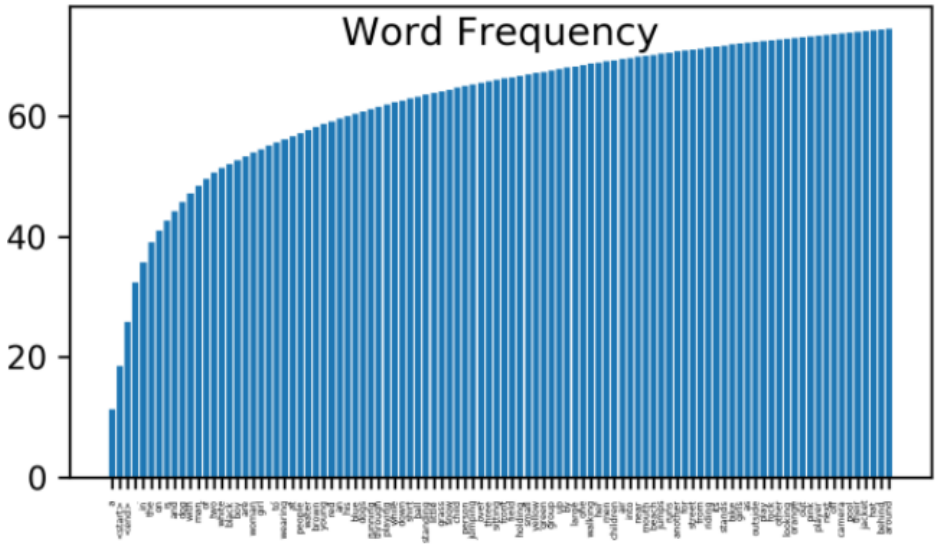

FIGURE 21 – la fréquence des mots utilisés

Nous constatons sur cette figure que la probabilité d'occurrence varie pour chaque mot ,en effet la probabilité d'occurrence des mots proche du centre sont ceux pour lequel la probabilité d'occurrence est faible , et les mots situant à la fin de l'axe sont pour ceux la probabilité d'occurrence est importante , avec un aspect lisse pour les mots situant à la fin .

Exercice 2 : Création des données d'apprentissage et de test

Nous allons maintenant stocker les données d'entraînement, i.e. les tenseurs contenant les données et les labels. Le tenseur des données X sera de taille Ns× Ls× d où Ns est le nombre de séquence (légendes), Ls est la longueur de la séquence et d est la taille du vecteur décrivant chaque mot de la séquence. Dans notre cas on a Ns=30000= 6000 images × 5 légendes par image, Ls=35 correspondant à la taille maximale d'une légende (calculé sur toute la base des légendes) et d=202 (100 embeddings vectoriels des images obtenus par l'extraction des deep feature par un réseau de type VGG puis par une réduction de dimension par acp, 102 embeddings vectoriels des légendes (En vrai il s'agit de 100 embeddings vectoriels et 2 pour les mots <start>, et <end>))

On crée ainsi notre base d'apprentissage et base de test (même dimension pour les deux)

La sortie du réseau récurrent est alors 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 "<end>". Le vocabulaire ayant été simplifié à l'exercice 1, on ne conservera dans les séquences d'entrée et de sortie que les mots de la légende présents dans le dictionnaire réduit.

Exercice 3 : Entraînement du modèle

Pour apprendre à prédire le mot suivant à partir d'une sous séquence données et d'une image , nous avons choisi une architecture du modèle pour l'entraînement comme suite :

- une couche de Masking avec une taille d'entrée de (35,202), pour ignorer les pas temporels car si une couche en aval ne prend pas en charge le masquage alors une exception sera levée.

- Une couche de réseau récurrent de type SimpleRNN avec 100 neurones dans la couche cachée qui renvoie une prédiction pour chaque élément de la séquence d'entrée .

- Une couche complètement connectée, suivie d'une fonction d'activation softmax.

l'apprentissage du modéle est effectué dans le but de minimiser la fonction de coût 'entropie croisé ' appliqué pour chaque élément de sortie ,avec l'utilisation d'un optimiseur ADAM ainsi qu'un pas de gradient de 0.001. La taille du batche est de 10 exemples (l'erreur de classification sur chaque légende est calculée comme une moyenne de prédictions sur chaque mot). que mot).

Après apprentissage de nos données d'entraînement. Les performances du modèle sur nos données test sont comme suit :

```
PERFS TRAIN: accuracy: 43.02%
PERFS TEST: accuracy: 41.26%
```

FIGURE 22 – Performances du Réseau

Nous constatons que le score obtenue est très faibles car le modèle n'arrive pas optimiser la fonction coût .

Exercice 4 : Évaluation du modèle

Finalement, on va utiliser le réseau récurrent pour générer une légende sur une image de test, et analyser qualitativement le résultat. On commence alors par charger un modèle appris, les données de test et les embeddings vectoriels avec le dictionnaire réduit. On sélectionne ensuite une image de l'ensemble test et on l'affiche ainsi que sa légende correspondante :

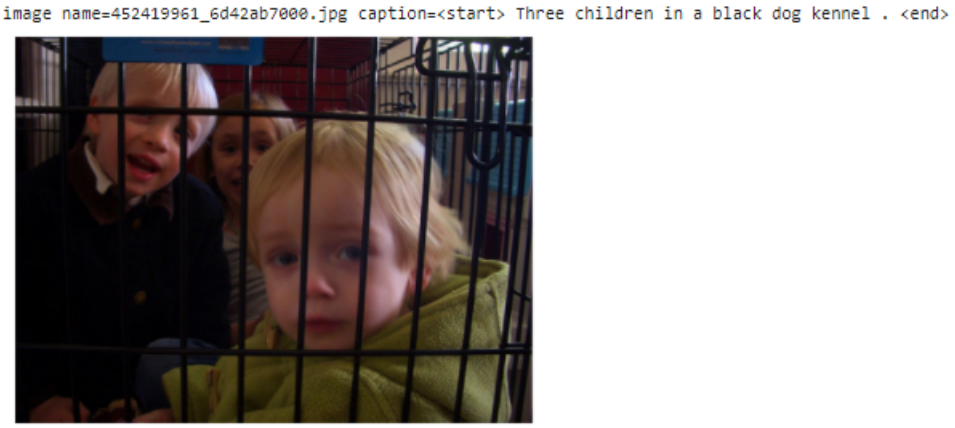

FIGURE 23 – Image sélectionnée et sa légende correspondante

Pour effecteur la prédiction, on va partir du premier élément de la séquence (i.e. contenant l'image et le symbole "<start>"), et effectuer la prédiction. On va ensuite pouvoir effectuer plusieurs générations de légendes, en échantillonnant le mot de suivant à partir de la distribution a posteriori issue du 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'à arriver au symbole de fin de séquence <"end">).

On obtient alors la séquence des prédictions suivantes :

L'évaluation du modèle se fait à travers la métrique du BLEU score (un nombre compris entre 0 et 1, où 0 représente zéro n-gramme de correspondance entre le texte prédit et le texte de référence et 1 pourrait être égal à une traduction automatique exactement similaire à l'une des références) Les résultats trouvés sont alors montrés par la figure suivante :

```
Yaml Model  Tp5_exo3 .yaml loaded
Weights  Tp5_exo3 .h5 loaded
157/157 [==============================] - 5s 22ms/step - loss: 0.9139 - accuracy: 0.4154
PERFS TEST: accuracy: 42.04%
i=0 ['a', 'brown', 'dog', 'is', 'running', 'through', 'the', 'snow', '.']
i=1000 ['a', 'girl', 'in', 'a', 'blue', 'dress', 'is', 'sitting', 'on', 'a', 'green', 'bench', '.']
i=2000 ['a', 'man', 'in', 'a', 'yellow', 'shirt', 'and', 'a', 'white', 'shirt', 'is', 'standing', 'on', 'a', 'large', 'bench', '.']
i=3000 ['a', 'young', 'boy', 'in', 'a', 'blue', 'shirt', 'is', 'jumping', 'a', 'fence', '.']
i=4000 ['a', 'young', 'boy', 'in', 'a', 'blue', 'shirt', 'is', 'playing', 'with', 'a', 'pool', '.']
blue_score - 0=0.5668402011823878
blue_score - 1=0.34102407335981794
blue_score - 2=0.20506189569398792
blue_score - 3=0.12338869898753847
```

FIGURE 24 – BLEU score du modèle

Le BLEU score trouvé varie entre 0.12 et 0.56 et donc notre modèle n'a pas une assez bonne performance quant à la prédiction des légendes de l'image, cependant il sera possible de l'améliorer si on augmente la taille de l'ensemble d'apprentissage, ou bien si l'on applique le modèle de "show attention and tell" où on étudie de plus près les différentes régions de l'image.

Bibliographie

[1] Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015) *Show and Tell : A Neural Image Caption Generator* . IEEE Xplore