TP 1 : Algorithme de rétropropagation de l'erreur

Objectif : L'objectif de cette première séance de travaux pratiques est d'implémenter par nous-même l'apprentissage des réseaux de neurones simples. Cette prise en main aura pour but d'approfondir nos connaissances et notre compréhension des réseaux de neurones simples. Elle permettra également de facilité notre compréhension du fonctionnement des librairies (comme Keras) où l'apprentissage est automatisé.

Au cours de ce TP nous travaillerons avec la base de données d'images MNIST, constituée d'images de caractères manuscrits (60000 images en apprentissage, 10000 en test).

Exercice 0 : Visualisation

Dans cette partie nous avons commencé par visualiser les 200 premières images de notre base de données.

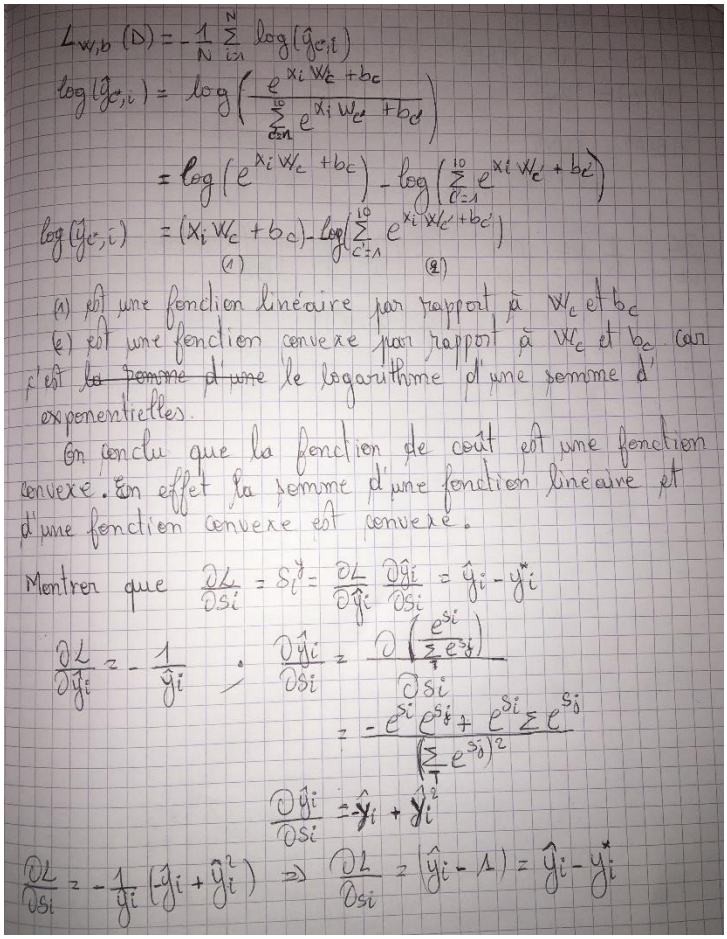
Les images se trouvent dans un espace à trois dimensions de taille (Nb_samples*28*28 où Nb_samples représente le nombre d'images)

Exercice 1 : Régression Logistique

Dans cet exercice, nous implémenterons et appliquerons (en utilisant les formules mathématiques) une régression logistique sur notre base d'apprentissage dans le but d'avoir un modèle capable de classifier les différents chiffres manuscrits.

La taille du vecteur d'entrée est 784 et celle du vecteur de sortie est 10, n'ayant pas ici de couches cachées alors le nombre de paramètres à estimer est égale à (784+1)*10 = 7850. Le 1 représentant les seuils à estimer.

Démonstrations :



Deep Learning avec Keras et Manifold Untangling

Objectif: L'objectif de cette partie est de prendre en main la librairie **Keras** pour utiliser et entraîner des réseaux de neurones profonds.

Introduction : Keras est une bibliothèque de Python pour le deep learning. Pour cette étude, on utilisera les données de la base MNIST récupérée de la bibliothèque **Keras.datasets**. Le travail sera présenté en deux partie, la première partie analyserais les différents modèles telle que : la régression logistique, perceptron multicouches (MLP) et les réseaux de neurones convolutifs. La seconde partie sera la visualisation des représentations internes des réseaux de neurones et des données de MNIST à l'aide de T-SNE et PCA afin d'illustrer la capacité des réseaux de neurones profonds à apprendre des représentations internes capables de résoudre le problème de « **Manifold untangling »** en neuroscience, c'est à dire de séparer les exemples des différentes classes dans l'espace de représentations appris.

Partie1 :

- **Regression Logistic avec Keras :**

Nous avons créé un réseau de neurone avec une couche de projection linéaire complètement connecter de taille 10, avec une couche d'activation de type softmax. Nous avons compilée notre modèle en utilisé la méthode d'optimisation de descente de gradient stochastique, en lui passant l'entropie croisée comme loss function et une métrique (de type 'accuracy') d'évaluation pour évaluer le taux de bonne prédiction des catégories. Nous avons utilisé un ensemble d'apprentissage de taille (60,000 784) et nous avons convertir les classe de vecteur en une matrice binaire. Nous avons utilisé 100 exemples pour estimer le gradient de la fonction de coût en faisant 20 passages sur l'ensemble des exemples de la base d'apprentissage lors de la descente de gradient.

Nous obtenons l'architecture suivante de notre modèle construit :

```
_________________________________________________________________
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
_________________________________________________________________
```

Nous avons donc entrainé notre modèle avec la méthode fit avec un pas d'entrainement de 0.1 et évalue les performances du réseau sur la base de test. Nous avons donc obtenue les résultats suivant :

- loss: **27.06%**

- acc: **92.34%**

La méthode d'optimisation par la descente de gradient stochastique nous permet d'éviter les minimaux locaux quand on essaye de converger vers le minimum global car cette méthode prend un pas en calculant le gradient de la fonction de cout pour un exemple aléatoirement sens remplacement.

- **Perceptron avec Keras**

Ici, nous enrichissons 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, ainsi nous allons obtenir un réseau de neurones à une couche cachée. Pour obtenir le perceptron, on ajoute à un réseau séquentiel vide une première couche cachée de taille 100 avec Keras, puis nous avons ajouté une autre couche de taille 10 pour obtenir la couche de sortie a 10 classes, et nous avons également ajouté une fonction d'activation de type soft-max. Nous avons compilée notre modèle en utilisé la méthode d'optimisation de descente de gradient stochastique, en lui passant l'entropie croisée comme loss function et une métrique (de type 'accuracy') d'évaluation pour évaluer le taux de bonne prédiction des catégories. Nous avons utilisé un ensemble d'apprentissage de taille (60,000 784) et nous avons convertir les classe de vecteur en une matrice binaire. Nous avons utilisé 100 exemples pour estimer le gradient de la fonction de coût en faisant 100 passages sur l'ensemble des exemples de la base d'apprentissage lors de la descente de gradient.

Le nombre de paramètres de notre modèle MLP devient (784+1)*100 + (100+1)*10 = 79510

Nous obtenons l'architecture suivante de notre modèle construit :

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
fc2 (Dense)                  (None, 100)               78500
_________________________________________________________________
activation_4 (Activation)    (None, 100)               0
_________________________________________________________________
fc3 (Dense)                  (None, 10)                1010
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0
=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
```

Nous avons donc entrainé notre modèle avec la méthode fit avec un pas d'entrainement de 1.0 et évalue les performances du réseau sur la base de test. Nous avons donc obtenue les résultats suivant :

Nous avons donc obtenue les résultats suivant :

- loss: **9.05%**
- acc: **97.95%**

Comparée aux perceptron de l'exercice précèdent nous avons une meilleurs performance et avec quelques lignes de codes.

- **Réseaux de neurones convolutifs avec Keras**

Ici on va maintenant étendre le perceptron précédent pour mettre en place un réseau de neurones convolutif profond, "Convolutionnal Neural Networks", ConvNets. Les réseaux convolutifs manipulent des images multi dimensionnelles en entrée (tenseurs). Alors, nous avons donc commencé par reformater les données d'entrée afin que chaque exemple soit de taille 28×28×1 (28 de large, 28 de haut, 1 canaux de couleur).

Par rapport aux réseaux complètement connectés, les réseaux convolutifs utilisent les briques élémentaires suivantes :

- Des couches de convolution, qui transforment un tenseur d'entrée de taille nx*ny*p en un tenseur de sortie nx′*ny′*nH, où nH est le nombre de filtres choisi.
- Des couches d'agrégation spatiale (pooling), afin de permettre une invariance aux translations locales.

Nous avons construire un modèle concolutive 2D avec 2 couche de convolution ainsi que 2 couches d'agrégation spatiale 2D (max-pooling) décrite comme suite :

- La première couche de convolution contient 32 filtres d'une taille spatiale de (5,5) qui correspond aux masque de convolution ; avec l'option padding='valid' pour ignorer les bords lors du calcul afin de diminuer la taille spatiale en sortie de la convolution ; une non-linéarité en sortie de la convolution de type relu. La seconde couche de convolution contient 16 filtres avec les mêmes caractéristiques que la première convolution.
- Les deux couches d'agrégations spatiales sont d'une taille de (2,2), sur lesquelles les opérations d'agrégation sont effectuées, elles permettent une invariance aux translations locales. Elles permettent d'obtenir des cartes de sorties avec des tailles spatiales divisées par deux par rapport à la taille d'entrée.

Notre modèle est donc construit en ajoutant une première couche de convolution, suivie d'une première couche max-pooling, suivie de la seconde couche de convolution, puis une seconde couche de max-pooling, puis nous avons mis à plat les couches convolution à l'aide de Flatten(). Nous avons ensuite ajouté une couche complètement connectée de taille 100, puis ajouter une non linéarité de type sigmoïde, nous avons encore ajouté une couche complètement connectée de taille 10, suivie d'une non linéarité de type softmax. L'architecture du modèle obtenu est comme suis :

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 24, 24, 32)	832
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 32)	0
conv2d_2 (Conv2D)	(None, 8, 8, 16)	12816
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 16)	0
flatten_1 (Flatten)	(None, 256)	0
fc4 (Dense)	(None, 100)	25700
activation_6 (Activation)	(None, 100)	0
fc5 (Dense)	(None, 10)	1010
activation_7 (Activation)	(None, 10)	0

Total params: 40,358
Trainable params: 40,358
Non-trainable params: 0

La première étape dans cette architecture est celle de la convolution de l'image que l'on souhaite classifier. La convolution est le fait de mettre en valeur quelques caractéristiques bien choisies dans l'image source afin d'avoir la même image mais avec une sorte de filtre. Cette opération est faite en appliquant une matrice prédéfinie sur la matrice source de l'image initiale. Plusieurs filtres de convolutions sont appliqués sur l'image initiale, conduisant donc à la génération de plusieurs images distinctes au niveau de la forme. Le principe est de faire "glisser" une fenêtre représentant la *feature* sur l'image, et de calculer le produit de convolution entre la *feature* et chaque portion de l'image balayée, une *feature* est alors vue comme un filtre.

La deuxième étape, dite pooling, consiste en la réduction des dimensions des images (matrices), elle reçoit en entrée plusieurs *feature maps*, et applique à chacune d'entre elles l'opération de ***pooling***. Le but étant de garder le maximum d'informations pertinentes même en réduisant les dimensions. La fonction de pooling utilisés dans notre cas est le Max-pooling qui fait partie des plus utilisé. On découpe l'image en cellules régulière, puis on garde au sein de chaque cellule la valeur maximale. Nous avons utilisé une cellule carrées de petite taille (2,2) qui ne se chevauchent pas, pour ne pas perdre trop d'informations. On obtient en sortie le même nombre de *feature maps* qu'en entrée, mais celles-ci sont bien plus petites. La couche de *pooling* permet de réduire le nombre de paramètres et de calculs dans le réseau, ainsi on améliore l'efficacité du réseau toute en évitant le sur-apprentissage. Les valeurs maximales sont repérées de manière moins exacte dans les *feature maps* obtenues après *pooling* que dans celles reçues en entrée. Ainsi, la couche de *pooling* rend le réseau moins sensible à la position des *features* : le fait qu'une *feature* ait une orientation légèrement différente ne devrait pas provoquer un changement radical dans la classification de l'image. Qui est en fait un grand avantage

Les étapes de Convolution et de Pooling sont ensuite itérées autant de fois que nécessaire jusqu'à ce que toutes les caractéristiques de l'image soit classifiée.

L'étape des couches complètement connectées est ajouté afin d'appliquée un réseau de neurones classique à toutes les caractéristiques classifiées par les étapes précédentes. Cette couche détermine

le lien entre la position des features dans l'image et une classe. La dernier couche complétement connectée permet de classifier l'image en entrée du réseau, elle renvoie un vecteur de taille représentant le nombre de classe dans le problème de classification de l'image, chaque élément du vecteur indique la probabilité pour l'image en entrée d'appartenir à une classe. L'output de ce réseau se réduit finalement à une probabilité qui permet de préciser à quel degré l'image en question est bien d'une classe précise, les valeurs élevées indiquent dans quelle classe l'image appartient. Puisque nous avons plusieurs classes (plus de deux) dans notre cas, nous appliquons une fonction d'activation de type **softmax**.

Nous avons compilée notre modèle en utilisé la méthode d'optimisation de descente de gradient stochastique, en lui passant l'entropie croisée comme loss function et une métrique (de type 'accuracy') d'évaluation pour évaluer le taux de bonne prédiction des catégories. Nous avons utilisé un ensemble d'apprentissage de taille (60,000 784) et nous avons convertir les classe de vecteur en une matrice binaire. Nous avons utilisé 100 exemples pour estimer le gradient de la fonction de coût en faisant 50 passages sur l'ensemble des exemples de la base d'apprentissage lors de la descente de gradient.

Nous avons donc entrainé notre modèle avec la méthode fit avec un pas d'entrainement de 1.0 et évalue les performances du réseau sur la base de test. Nous avons donc obtenue les résultats suivant :

Nous avons donc obtenue les résultats suivant :

- loss: **3.14%**
- acc: **99.12%**
- **Avec un temps d'epouche 1ms/step**

Nous remarquons une amélioration avec la méthode de convolution. Nous devions apprendre le modèle en utilisant une carte GPU pour comparer le temps d'un epouche mais par manque de GPU nous n'avons pas pu le faire. Mais pour la question qui est d'où vient ce gain en temps d'une epouche, les recherches nous indiques qu'il vient du fait que le calcule est faite en fessant usage du parallélisme du GPU. Le GPU permet d'exécuter les taches parallèles (comme par exemple multiplier les matrices) plus rapidement, ceci est dû à son architecture. Nous obtenons un gain en temps pendant le training car TensorFlow ne fait que multiplier des matrices derrière, et sur un GPU c'est fait de façon parallèle.

Partie2 :

Le but de cette partie est d'illustrer la capacité des réseaux de neurones profonds à apprendre des représentations internes capables de résoudre le problème connu sous le nom de « manifold untangling » en neuroscience, c'est à dire de séparer les exemples des différentes classes dans l'espace de représentation appris. Nous allons utiliser des outils de visualisation qui vont permettre de représenter chaque donnée 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.

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 présentent des positions 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.

On va appliquer la méthode t-SNE sur les données brutes **de la base de test de MNIST** en utilisant la classe T-SNE du module sklearn.manifold. Nous allons effectuer une réduction de dimension en 2D des données de teste de la base MNIST en utilisant la méthode **T-SNE** et la méthode **d'ACP.**

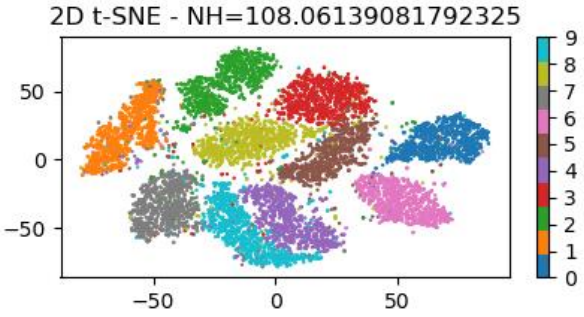
Afin de visualiser l'ensemble des points projetés en 2D, nous avons définir des critères pour analyser la séparabilité des classes dans l'espace projeté.

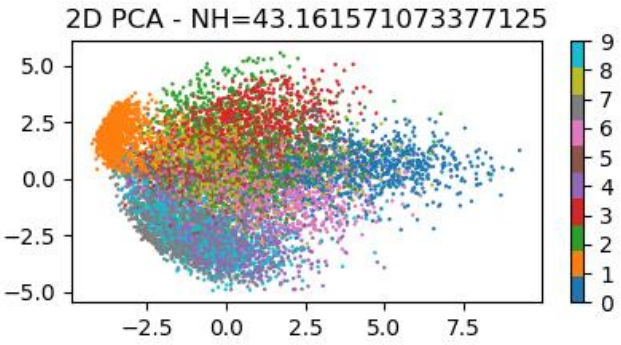
- l'enveloppe convexe des points projetés pour chacune des classes avec la classe **ConvexHull** du module **scipy.spatial**.
- l'ellipse de meilleure approximation des points avec la classe **GaussianMixture** du module **sklearn.mixture**.
- Et le "Neighborhood Hit" (NH) avec la class **NearesrNeighbors** du module **sklearn.neighbors.**

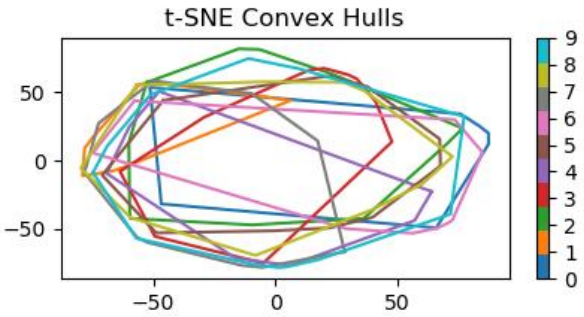
Les trois métriques ci-dessus sont-elles liées au problème de la séparabilité des classes ? Oui elles le sont :

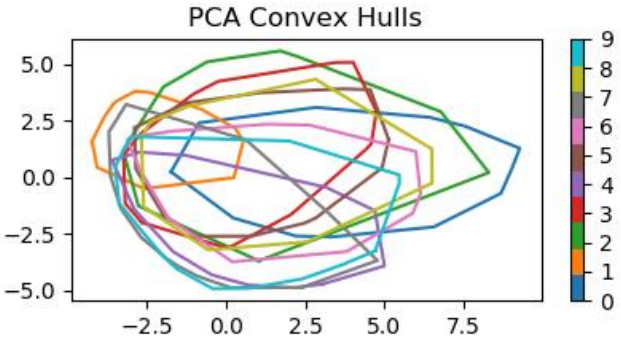
Le concept général de l'algorithme t-sne est de considérer chaque point de données séparément, et d'assigner une probabilité conditionnelle (un poids) gaussienne à chacun des autres points en fonction de leur **distance** par rapport à ce point. L'idée est de trouver un espace de dimension inférieure mais qui respecte une distribution proche en divergence de la distribution d'origine. La probabilité met l'accent sur la structure locale, puisqu'elle met plus de poids sur les petites distances entre les points du jeu de données.

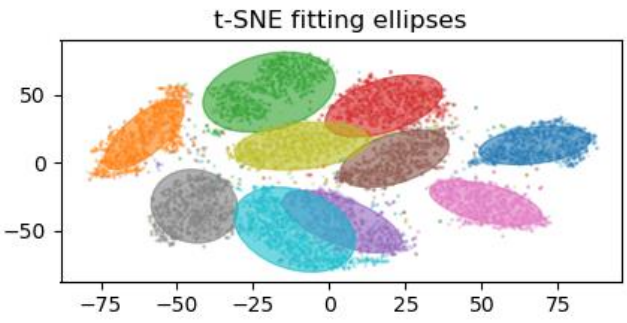
On remarque que le t-sne sépare bien les classes.

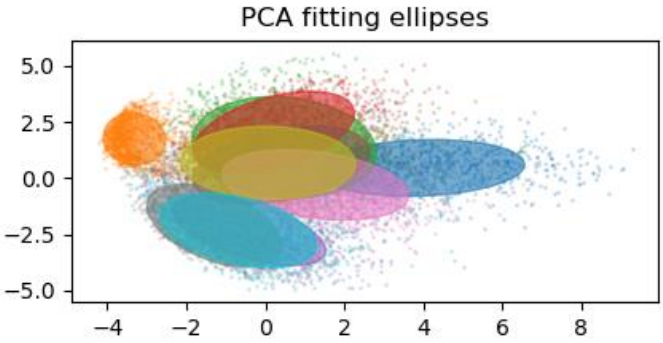






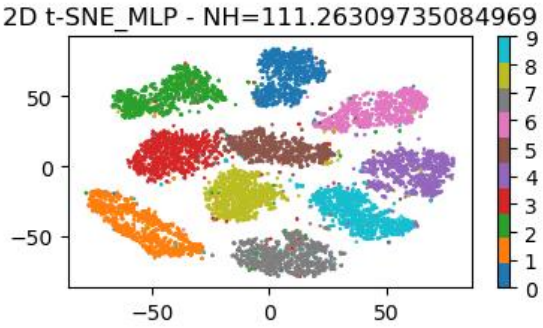


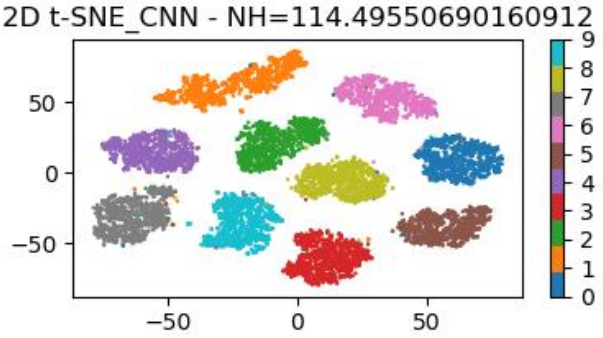


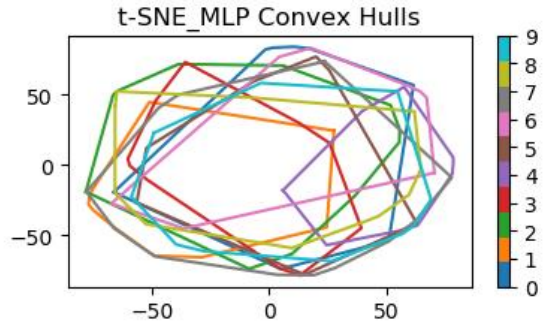


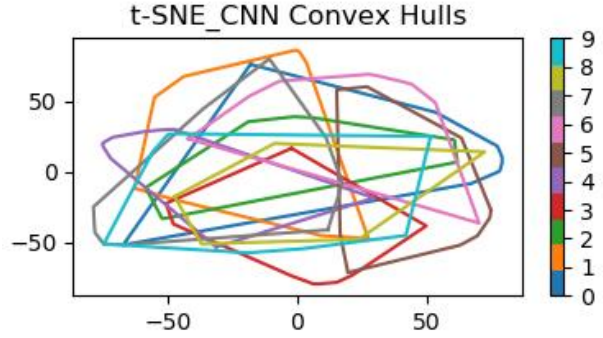
- **Visualisation des représentations internes des réseaux de neurones**

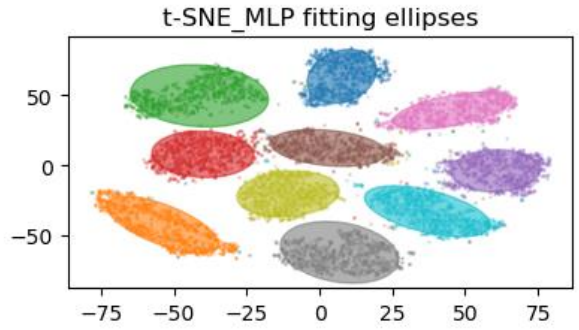
Ici nous avons rechargé les modèle MLP et ConvNet afin de visualiser les prédictions. Nous avons supprimé les couches au sommet des modèles (la couche d'activation softmax et la couche complètement connectée). Nous obtenons les visualisations avec un Perceptron (à gauche) et un réseau convolutif (à droite) suivantes :

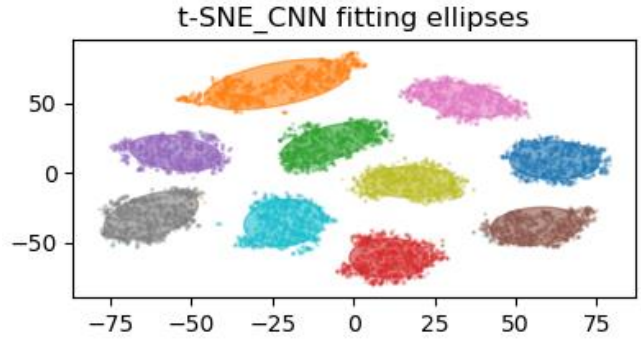












On remarque que la méthode de convolution a bien séparé les classes que le perceptron. Les deux méthodes on aussi mieux séparer que précédemment.

TP 3 - Transfer Learning et Fine-Tuning

Pour ce TP nous allons nous intéresser aux propriétés de "transfert" des réseaux convolutifs profonds pré-entraînés sur des bases large échelle comme ImageNet.

Exercice 1 : Modèle ResNet-50 avec Keras

Le principe du Transfer Learning est l'utilisation des connaissance acquises en résolvant pour résoudre un problème similaire. On va ici s'intéresser aux réseaux ResNet [HZRS16] donnant de très bonnes performances sur ImageNet et qui ont remporté le challenge ILSVRC en 2015. Nous utiliserons ce modèle sur la base PASCAL VOC 2007 afin d'effectuer une classification (mais ici avec 20 classes)

Exercice 2 : Extraction de "Deep Features

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".

Chargement des données de la base : le stockage en mémoire de l'intégralité de la base PASCAL VOC 2007 où le tenseur d'entrée prend plusieurs Go de mémoire 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".

Exercice 3 : Transfert sur VOC 2007

Après l'extraction des Deep Features nous allons maintenant les considérer comme les données d'entrée et définir un réseau de neurones complètement connectés sans couche cachée pour prédire les labels de sortie.

Nous souhaitons ici avoir en sortie un vecteur de taille 20x1 définit avec un encodage one-hot représentant nos classes. La fonction sigmoïde permet de retourner pour les neurones de sorties (représentant chacune une classe) 1 si la classe est détectée dans l'image 0 sinon. La fonction softmax nous aurait retourné pour chacune des classes sa probabilité d'apparition, il aurait ensuite fallu appliquer une autre transformation pour obtenir un vecteur de sortie avec l'encodage « one-hot ».

Binary_crossentropy : On calcule la moyenne de l'entropie croisée de la manière suivante :

$H(q) = -(\frac{1}{N}) \sum_{c=1}^{N} q(y_c).\log(p(y_c))$, avec $q(y_c)$ probabilité réelle de la classe c, $p(y_c)$ la probabilité prédite de la classe c et N le nombre de classes.

Pour chaque image $q(y_c)$=1 si le label est présent et 0 sinon. On constate donc que plus la probabilité prédite de la classe est proche de 1 plus l'entropie croisée est proche de 0 et on obtient un bon classifieur.

Le binary_crossentropy est adaptée au contexte multi-label dans le sens où N le nombre de classe dans le calcul de l'entropie croisée peut être supérieur ou égale à 2.

Après avoir entraîné et évalué le modèle classiquement on obtient sur la base de test un MAP de 82.81% avec les paramètres suggéré lors du lors du TP, mais en variant les paramètres nous avons légèrement obtenu sur la base de test un meilleur MAP de 83% avec 50 epoch et en gardant lr= 0.1.

Exercice 4 : Fine-tuning sur VOC 2007

Le Fine Tuning est un moyen d'application ou d'utilisation du Transfer Learning. On utilise un modèle qui a été entrainé pour une tâche donnée et on le fine-tune (modifie) pour l'adapter et l'appliquer sur un problème similaire.

Si on avait indiqué model.layers[i].trainable = False pour toutes les couches sauf la dernière, on serait dans le cas du Transfer Learning.

Le Transfer Learning et le Fine Tuning permettent d'avoir un gain, lors de l'apprentissage concernant le choix du learning rate, de la complexité du modèle,… et de se servir directement des connaissances acquises précédemment sur un cas similaire.

TP 4 - Réseaux de neurones récurrents

L'objectif de ce TP est d'utiliser des réseaux de neurones récurrents pour l'analyse de données séquentielles.

Exercice 1 : Génération de poésie

A partir d'une base de données d'un recueil de poésies, "les fleurs de mal" de Charles Baudelaire et après avoir effectué quelques prétraitements nous avons constitué un ensemble d'apprentissage et de test formés chacun de séquences de longueur fixe (10) labelisées chacune avec une étiquette cible correspondant au prochain caractère à prédire.

Analyse de l'apprentissage

Après avoir entrainé notre réseau de neurones on a obtenu une performance assez basse en apprentissage de l'ordre de 53.53%.

Les réseaux de neurones récurrents classiques sont exposés au problème de disparition de gradient qui les empêchent de modifier leur poids en fonction d'évènements passés. En effet, les RNNs n'arrivent pas à capturer les dépendances longues distances entre les mots car les gradients dans les instances de longues séquences ont une grande chance soit d'être réduit à 0 ou de tendre vers l'infini très rapidement. Quand les gradients tombent rapidement à 0 le modèle n'est pas capable d'apprendre les associations ou corrélations entre les évènements qui sont temporairement éloignés.

La différence majeure entre le problème de classification abordé ici et les problèmes de classifications vus jusqu'ici est qu'ici nous travaillons avec des données séquentielles et que cela nécessite la prise en compte des séquences (états) précédentes pour un instant t donné.

Dans les réseaux récurrents l'information peut se propager dans les deux sens, y compris des couches profondes aux premières couches. Les RNNs 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. Pour cette raison, les RNNs sont particulièrement adaptés aux applications faisant intervenir le contexte, et plus particulièrement au traitement des séquences temporelles.

Génération de texte avec le modèle appris

Dans cette partie nous avons initialisé une chaîne de caractère pour notre réseau, afin de prédire le caractère suivant et ainsi de générer une chaine de caractères. 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.

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.

Analyse de la génération

En faisant varier le nombre d'époques nous n'avons pas obtenus (remarqués) d'améliorations sur la génération du texte. Par contre la variation du paramètre « température » a eu un impact dans la génération de la séquence de texte.

A partir de T>=0.5 on commence à remarquer que notre générateur de texte génère de plus en plus de mots qui n'ont pas de sens.

Pour T = 0.1 on obtient un assez grand nombre de mots qui ont du sens et leur agencement également ont parfois du sens.

Cependant pour T petit de l'ordre de 0.01 on a des mots qui ont du sens et leur agencement également à du sens, mais notre générateur tend à générer les mêmes mots.

Le générateur de texte s'il fonctionne plutôt bien dois encore être améliorer. Pour cela il faudra entrainer notre modèle avec une base de données plus grande.

Exercice 2 : Embedding Vectoriel de texte

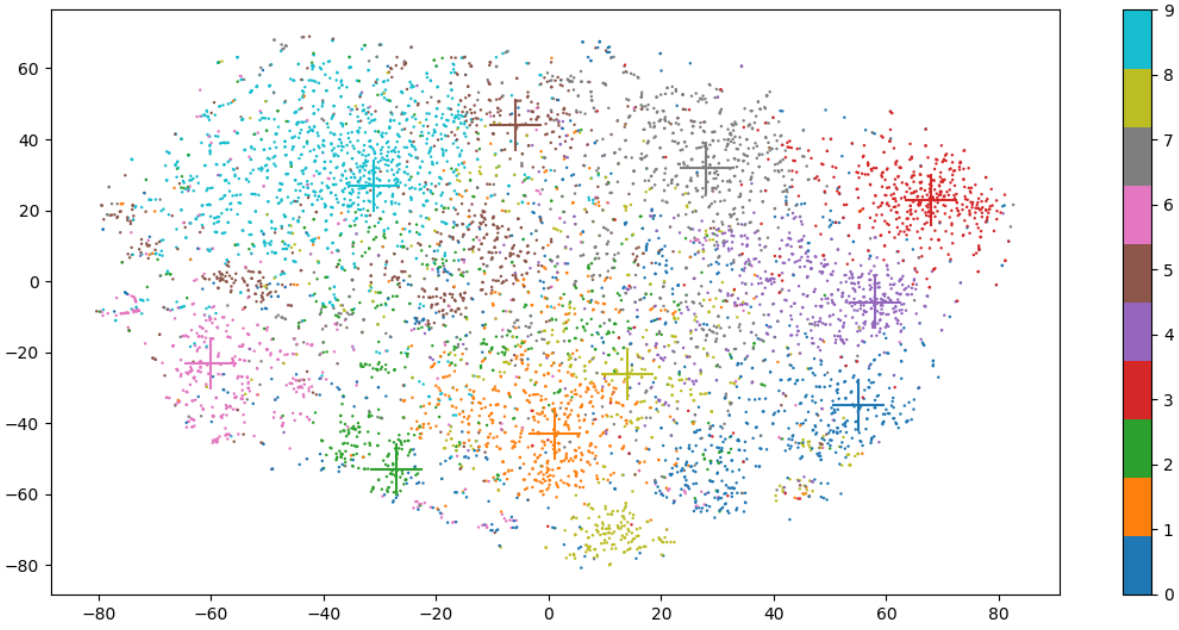
Dans cet exercice, nous allons explorer l'embedding vectoriel de texte Glove [PSM14]. On va donc utiliser la base d'image FlickR8k pour laquelle chaque image est associée à 5 légendes différentes qui décrivent son contenu en langage naturel.

La base d'apprentissage contient 6000 images, ce qui correspond à 30000 légendes.

Analyse des embedding Glove des légendes

Après avoir observé le résultat des centres du clustering obtenu ainsi que les mots les plus proches de chaque centre on constate que les mots proches de chaque centre se trouve dans le même champ lexical que le mot référent (centre du cluster).

Ci-dessous une visualisation de la répartition des points dans l'espace d'embedding :

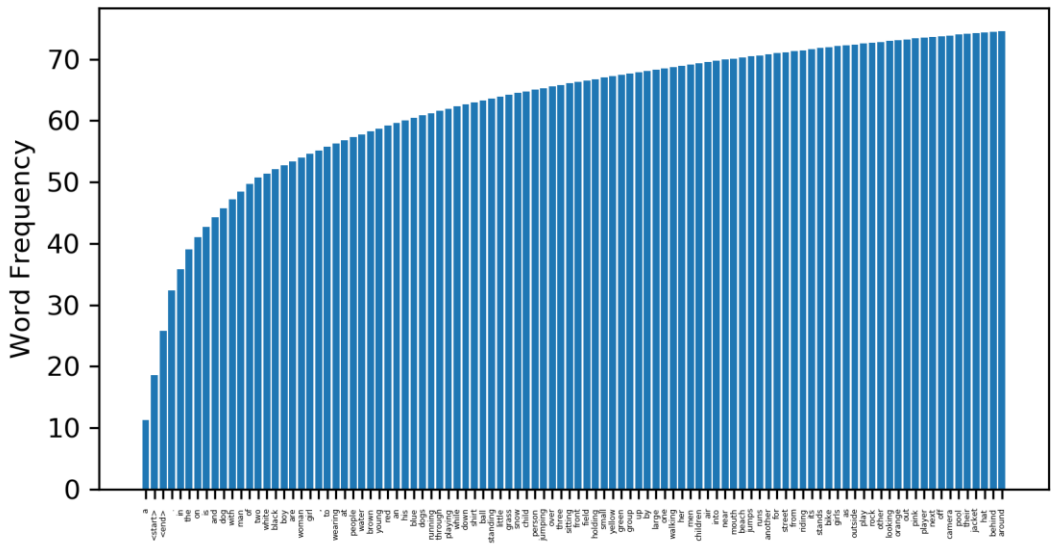


TP 5 - Vision et language

Objectif: L'objectif de cette partie est d'aborder le problème du légendage d'images, qui consiste à décrire le contenu visuel d'une image par une phrase en language naturel. Nous allons mettre en place une version simplifiée de l'approche "show and tell".

Introduction : 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>').

Pour accélérer le temps nécessaire à l'entraînement du modèle, nous allons considérer un sous-ensemble du vocabulaire de mots considéré au TP précèdent. La fréquence cumulée des 100 premiers mots conservés, nous donne la figure suivante :



Nous avons séparé nos données en ensemble de train de taille (10000, 784). Les tenseurs contenant les données et les labels. Le tenseur des données X sont 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, et d est la taille du vecteur décrivant chaque mot de la séquence.

Les fichiers sources utilisé contiennent les identifiants des images et les légendes des images, la longueur de la légende maximale dans les données d'entraînement étant de

35. Chaque élément ei d'une séquence d'entrée va être décrit par un vecteur xi∈Rd (avec ici d=202) correspondant à :

- La description du contenu du ie`me mot de la légende, pour laquelle on utilisera l'embedding vectoriel Glove. Ceci correspondra aux 102 premières composantes de xi.
- 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 dimension du vecteur résultant (ici 4096) a été réduite à 100 par Analyse en Composantes Principales (ACP). Ceci correspondra aux 100 dernières composantes de xi. 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 termine toujours par '<end>'. Le vocabulaire ayant été simplifié, on ne conservera dans les séquences d'entrée et de sortie que les mots de la légende présents dans le dictionnaire réduit.

La même chose a été faite pour les données de test, en allouant des tenseurs de la même taille que ceux d'entraînement (et pouvoir ainsi appliquer le modèle de prédiction ensuite).

Entraînement du modèle

L'architecture du modèle pour apprendre à prédire le mot suivant à partir d'une sous-séquence donnée et d'une image a été définie comme suit :

Nous avons commencé par instancier un modèle vide puis ajouter:

- Une couche de Masking qui permettre de ne pas calculer d'erreur dans les zones où le tenseur d'entrée sera à une valeur 0, ce qui correspondra au zones de la séquence d'entrée où aucun mot n'est présent (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 simple avec 100 neurones dans la couche cachée, en renvoyant les prédictions pour chaque élément de la séquence d'entrée.
- Une couche complètement connectée de taille 1000, suivie d'une fonction d'activation softmax.
- Nous avons utilisé 10 exemples pour estimer le gradient de la fonction de coût en faisant 10 passages sur l'ensemble des exemples de la base d'apprentissage lors de la descente de gradient.

Nous avons compilée notre modèle en utilisé la méthode d'optimisation de Adam, en lui passant l'entropie croisée comme loss function et une métrique (de type 'accuracy') d'évaluation pour évaluer le taux de bonne prédiction des catégories.

Notre modèle a l'architecture suivante :

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
masking_3 (Masking)          (None, 35, 202)           0
_________________________________________________________________
simple_rnn_3 (SimpleRNN)     (None, 35, 100)           30300
_________________________________________________________________
fc5 (Dense)                  (None, 35, 1000)          101000
_________________________________________________________________
activation_3 (Activation)    (None, 35, 1000)          0
=================================================================
Total params: 131,300
Trainable params: 131,300
Non-trainable params: 0
```

Évaluation du modèle

Pour évaluer le modèle, nous avons utilisé le réseau récurrent pour générer une légende sur une image de test, et analyser qualitativement le résultat. D'abord, notre modèle testé sur train et test nous donne les performances suivant :

- loss TRAIN: **270.93%**
- acc TRAIN: **41.42%**
- loss TEST: **282.90%**
- acc TEST: **40.11%**

Nous avons ensuite sélectionné une image parmi l'ensemble de train, l'afficher, ainsi qu'une légende issues des annotations et nous avons obtenu le résultat suivant :

image name=1433577867_39a1510c43.jpg caption=<start> A white dog treads the water with sticks in its mouth . <end>

Nous avons ensuite sélectionné une image parmi l'ensemble de train, l'afficher, ainsi qu'une légende issues des annotations et nous avons obtenu le résultat suivant :

image name=127488876_f2d2a89588.jpg caption=<start> Two golfers are standing on the fairway looking off into the distance . <end>

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 suivant à partir de la distribution a posteriori issue du softmax. Ont obtiens les résultats suivants pour la dernier image:

Caption n° 1: a man in a red shirt is sitting on a bench in a park . <end>

Caption n° 2: a man in a red shirt is sitting on a bench in a park . <end>

Caption n° 3: a boy in a red shirt is sitting on a bench in a park . <end>

Caption n° 4: a man in a red shirt is sitting on a bench in a park . <end>

Caption n° 5: a man in a red shirt is sitting on a bench . <end>

On remarque que les légende obtenu est un peu raisonnable car sa identifie aux moins un homme et red shirt mais ils sont debout plutôt.

Blue Score :

```
Yaml Model  model_10mil .yaml loaded
Weights  model_10mil .h5 loaded
5000/5000 [==============================] - 15s 3ms/step
PERFS TEST: acc: 40.11%
i=0 ['a', 'brown', 'dog', 'is', 'running', 'through', 'the', 'grass', '.']
i=1000 ['a', 'little', 'girl', 'is', 'sitting', 'on', 'a', 'bed', '.']
i=2000 ['a', 'man', 'in', 'a', 'white', 'shirt', 'is', 'standing', 'on', 'a',
'beach', '.']
i=3000 ['a', 'man', 'in', 'a', 'black', 'shirt', 'and', 'white', 'pants', 'and',
'black', 'pants', 'and', 'a', 'white', 'shirt', 'and', 'black', 'pants', 'is',
'running', 'through', 'the', 'grass', '.']
i=4000 ['a', 'little', 'girl', 'in', 'a', 'pink', 'shirt', 'is', 'jumping', 'into',
'a', 'pool', '.']
blue_score - 0=0.5305932990125749
blue_score - 1=0.31361155880785063
blue_score - 2=0.1842137765218392
blue_score - 3=0.10805810770630063
```

Avec un accuracy de **40%** en test, notre première blue score nous donne **53%** ceci dit que les mots de la vraie légende et les mots prédit se correspondent un peut, sa explique le fait que nous avons aux moins des mots dans la légende qui représentes l'image. Par exemple a man in red shirt. Ces résultats sont pour la dernière image présente, des deux hommes.