Auto-encodeurs avec Keras

Mots clés :

- Auto-encodeurs
- Auto-encodeurs à convolutions
- Auto-encodeurs débruiteur

Jeu de données support : alphabet NotMNIST comme exemple.

Auto-encodeur

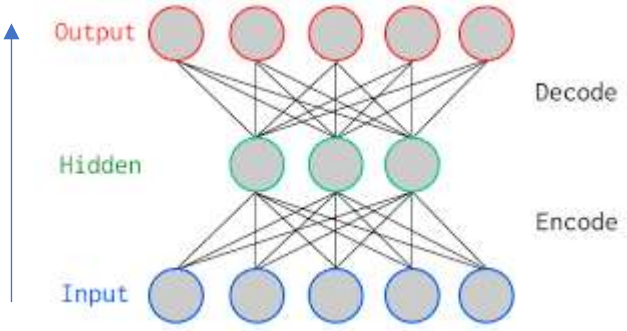
Les auto-encodeurs représentent une technique d'apprentissage non supervisée, les données d'entrainement ne sont pas labellisées.

Un auto-encodeur reçoit une donnée en entrée et tente de la reconstruire en utilisant le moins de bits possible au goulot d'étranglement également connu sous le nom d'espace latent. La donnée est majoritairement compressée au goulot d'étranglement.

Les auto-encodeurs sont similaires aux techniques de réduction de la dimensionnalité comme l'analyse des composants principaux (PCA). Ils projettent les données en réduisant leurs dimensions par transformation linéaire en essayant de préserver les caractéristiques importantes et en supprimant les parties non essentielles.

Cependant, la principale différence entre les auto-encodeurs et la PCA réside dans la partie de transformation : PCA utilise la transformation linéaire alors que les auto-encodeurs utilisent des transformations non linéaires.

La figure ci-dessous est un auto-encodeur à deux couches avec une couche cachée.

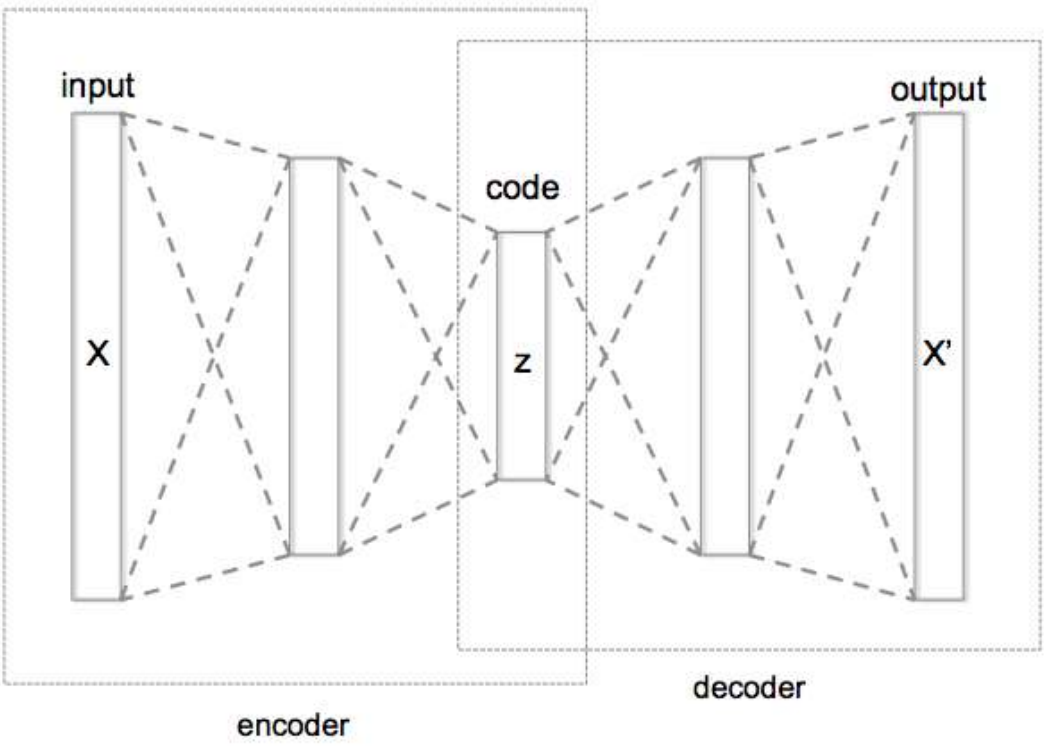


On remarque que les couches d'entrée et de sortie ont le même nombre de neurones.

Exemple :

- L'image d'entrée est codée avec seulement cinq pixels.
- Elle est compressée par l'encodeur en trois pixels au niveau du goulot d'étranglement (couche intermédiaire) ou de l'espace latent.
- À l'aide de ces trois valeurs, le décodeur essaie de reconstruire les cinq pixels ou plutôt l'image d'entrée

Dans la réalité, il y a plus de nombre de couches cachées entre l'entrée et la sortie.



L'auto-encodeur peut être divisé en trois parties

Encodeur :
Cette partie du réseau compresse ou minimise l'entrée. L'espace représenté par ces données compressées est souvent appelé l'espace latent ou le goulot d'étranglement. Ces données compressés qui représentent l'entrée d'origine sont appelés un « codage » de l'entrée.

Décodeur:
Cette partie du réseau essaie de reconstruire l'entrée en utilisant uniquement le codage de l'entrée. Lorsque le décodeur est capable de reconstruire l'entrée exactement comme il a été alimenté à l'encodeur, on dit qu'on a obtenu les meilleurs encodages pour l'entrée.

Auto-encodeurs à convolutions en Python avec keras

Étant donné que vos données d'entrée se composent d'images, il est judicieux d'utiliser un auto-encodeur à convolution.
Le principe de construction est simple : on remplace les couches denses par des couches à convolution. Les couches de convolution, convertissent l'entrée (28 x 28 sur un canal unique ou échelle de gris) en une plus petite (7 x 7 sur 128 canaux à l'espace latent).

C'est le sous-échantillonnage : processus dans lequel l'image se comprime dans une dimension basse également connue sous le nom d'encodeur.

Cela permet au réseau d'extraire des entités visuelles des images et donc d'obtenir une représentation plus précise de l'espace latent.

Le processus de reconstruction utilise le suréchantillonnage et les circonvolutions qui sont connus comme un décodeur.

Chargement des données

Le jeu de données notMNIST est un jeu de données de reconnaissance d'image de glyphs de police pour les lettres A à J. Il est tout à fait similaire au jeu de données MNIST classique, qui contient des images de chiffres manuscrits de 0 à 9 : le jeu de données notMNIST comprend des images 28x28 en niveaux de gris de 70 000 lettres de A à J au total 10 catégories, et 7 000 images par catégorie.

Prétraitement des données

Les données sont fournies au format csv : train_data.csv , test_data.csv, train_lables.csv et test_labels.csv

Dans les jeux de train et test les images sont aplaties (une image par ligne)

Il faut d'abord convertir chaque image 28 x 28 du jeu d'entraînement et du jeu de test dans une matrice de taille 28 x 28 x 1. De même, les labels doivent être au format numpy array 1D.

```
train_data.shape, test_data.shape
((60000, 28, 28, 1), (10000, 28, 28, 1))

train_labels.shape, test_labels.shape
((60000,), (10000,))
```

Il faut ensuite standardiser les valeurs entre 0 et 1

Construction de l'auto-encodeur à convolution

On utilisera une taille de lot (batch size) de 128.

Le réseau sera entrainé sur 50 Epochs.

Architecture de l'encodeur

La première couche aura 32 filtres de taille 3 x 3, suivis d'une couche de sous-échantillonnage (MaxPooling2D),

La deuxième couche aura 64 filtres de taille 3 x 3, suivis d'une autre couche de sous-échantillonnage,

La dernière couche de codeur aura 128 filtres de taille 3 x 3.

Architecture du décodeur

La première couche : 128 filtres -3 x 3 suivis d'une couche de suréchantillonnage,

La deuxième couche : 64 filtres - 3 x 3 suivis d'une autre couche de suréchantillonnage, (upSampling2D)

La dernière couche : 1 filtre - 3 x 3.

Remarque : le nombre de filtres, la taille du filtre, le nombre de couches, le nombre d'epochs que vous entraînez votre modèle, sont tous des hyperparamètres et doivent être décidés en fonction de votre propre intuition, vous êtes libre d'essayer de nouvelles expériences en peaufinant avec ces hyperparamètres et de mesurer performances de votre modèle. Et c'est ainsi que vous apprendrez lentement l'art de l'apprentissage profond !

Compilation

Une fois le modèle créé, vous devez le compiler à l'aide de l'optimiseur pour être RMSProp (Tutoriel par Andrew ng sur RMSProp) : https://fr.coursera.org/lecture/deep-neural-network/rmsprop-BhJlm

La fonction de cout (loss) sera l'erreur quadratique moyenne, puisque la perte après chaque lot sera calculée entre le lot de la sortie prévue et celle obtenue.

Summary :

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	(None, 28, 28, 1)	0
conv2d_7 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_3 (MaxPooling2	(None, 14, 14, 32)	0
conv2d_8 (Conv2D)	(None, 14, 14, 64)	18496
max_pooling2d_4 (MaxPooling2	(None, 7, 7, 64)	0
conv2d_9 (Conv2D)	(None, 7, 7, 128)	73856
conv2d_10 (Conv2D)	(None, 7, 7, 128)	147584
up_sampling2d_3 (UpSampling2	(None, 14, 14, 128)	0
conv2d_11 (Conv2D)	(None, 14, 14, 64)	73792
up_sampling2d_4 (UpSampling2	(None, 28, 28, 64)	0
conv2d_12 (Conv2D)	(None, 28, 28, 1)	577

Total params: 314,625
Trainable params: 314,625
Non-trainable params: 0

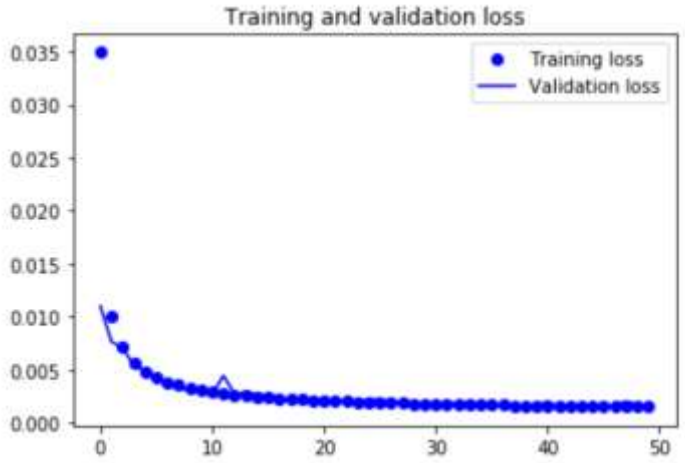
Entrainement

Utiliser la fonction fit () de Keras ! Le modèle s'entraînera sur 50 epochs.

Stocker l'objet history retourné par fit (). Il servira à tracer l'évolution de la fonction de coût d'entraînement et de validation pour analyser les performances du modèle.

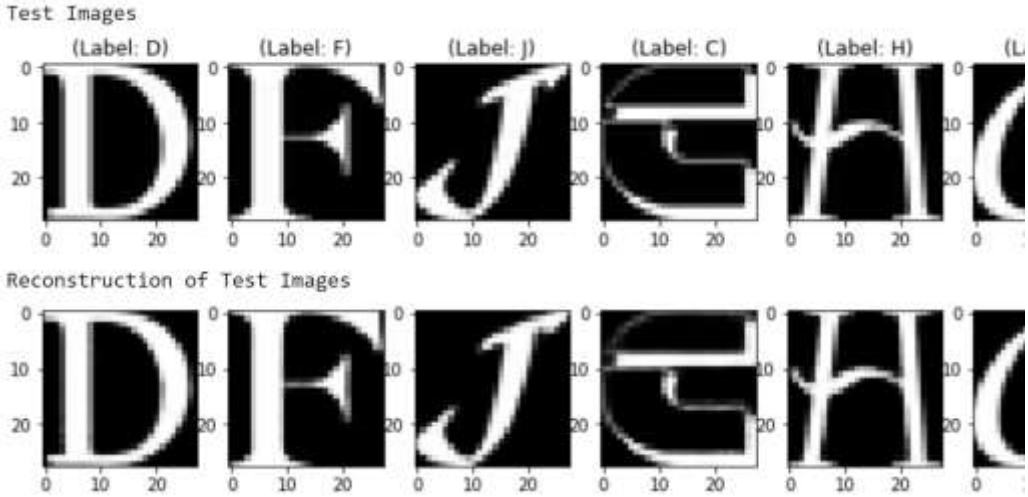
Visualisation de l'amélioration du modèle

Tracer le diagramme de perte entre les données d'entrainement et de validation pour visualiser les performances du modèle.



Prédiction sur les données de test

Prédire le modèle formé sur l'ensemble des 10 000 images de test et tracer quelques-unes des images reconstruites pour visualiser la façon dont votre modèle est capable de reconstruire les images.



Auto-encodeur débruiteur

Un auto-encodeur débruiteur essaie d'apprendre une représentation (espace latent ou goulot d'étranglement) qui est robuste au bruit. Vous ajoutez du bruit à une image, puis vous alimentez l'image bruyante en tant qu'entrée de la partie encoder de votre réseau. La partie encodeur de l'auto-encodeur transforme l'image en un espace différent qui tente de préserver les alphabets mais supprime le bruit.

How ?

Pendant l'entraînement, vous définissez une fonction de perte, semblable à l'erreur quadratique moyenne de la racine que vous aviez définie précédemment dans l'auto-encodeur à convolution. À chaque itération de la formation, le réseau calculera une perte entre l'image bruitée émise par le décodeur et la vraie image et tentera également de minimiser cette perte ou différence entre l'image reconstruite et l'originale sans bruit.

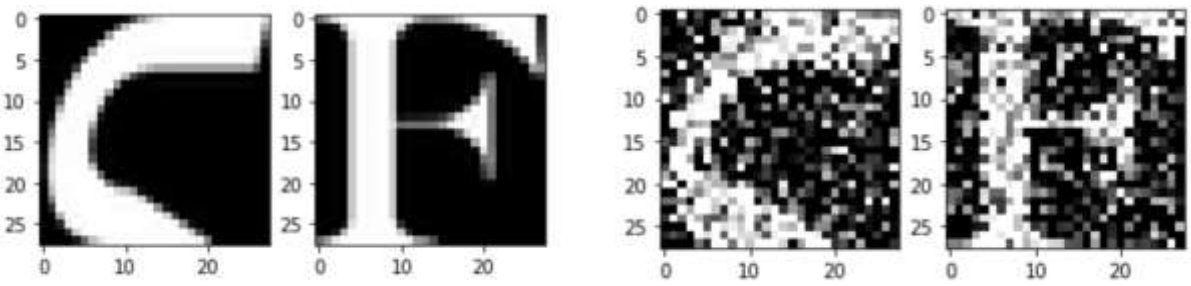
Mise en œuvre de l'Auto-encodeur débruiteur

Ajout de bruit aux images

Définir d'abord un facteur de bruit qui est un hyperparamètre.
Le facteur de bruit est multiplié par une matrice aléatoire qui a une moyenne de 0,0 et un écart-type de 1,0. Cette matrice va contenir des échantillons de la distribution normale (gaussienne). La dimension du tableau normal aléatoire sera similaire à celle des données auxquelles vous ajouterez le bruit.

Exemple de résultat :

Les deux premières images des jeux d'entrainement et de test



Architecture

Encodeur

La première couche : 32 filtres 3x 3 suivis d'une couche de sous-échantillonnage (MaxPooling2D),
La deuxième couche : 64 filtres 3x 3 suivis d'une autre couche de sous-échantillonnage,
La dernière couche d'encodeur : 128-3 filtres x 3.

Décodeur

La première couche : 128 filtres 3 x 3 suivis d'une couche de suréchantillonnage, (UpSampling2D)
La deuxième couche : 64 filtres 3 x 3 suivis d'une autre couche de suréchantillonnage,
La dernière couche d'encodeur : 1 filtre 3 x 3.

Entrainement

Dans l'autoencodeur débruiteur, l'entrée est constituée d'images bruitées tandis que la sortie attendue reste les images originales (sans le bruit). Le calcul de perte se fait donc entre les images débruitées reconstruites et les véritables images.

Prédiction

Faire une prédiction sur le jeu de test bruité et afficher 10 images de chaque groupe : réelles, bruitées et prédites.