RAPPORT INFO0809
PROJET TER

Création d’auto-encodeurs avec ANN & CNN

Réalisé par :
Maxence FUZELLIER

Table des matières

Introduction

Dans le cadre du projet TER, il m'a été demandé de réaliser deux auto-encodeurs capables de reconstruire des données véhiculaires (signaux 1D), le premier basé sur des réseaux de neurones artificiels entièrement connectés, puis le second sur des réseaux de neurones à convolution.

Outils et environnement de travail

Jupyter Notebook

Jupyter Notebook est une application client-serveur créée par l'organisation à but non lucratif Project Jupyter. Elle a été publiée en 2015. Elle permet la création et le partage de documents Webauformat JSON constitués d'une liste ordonnée de cellules d'entrées et de sorties et organisés en fonction des versions successives du document. Les cellules peuvent contenir, entre autres, du code, du texte au format Markdown, des formules mathématiques ou des contenus médias (Rich Media). Le traitement se fait avec une application client fonctionnant par Internet, à laquelle on accède par les navigateurs habituels. Il est nécessaire pour cela que soit installé et activé dans le système le serveur Jupyter Notebook. Les documents Jupyter créés peuvent s'exporter aux formats HTML, PDF, Markdown ou Python par exemple, ou bien se partager par email, avec Dropbox, GitHub ou un lecteur Jupyter Notebook.

Les deux éléments principaux de Jupyter Notebook sont un jeu de différents noyaux (interpréteurs) et le tableau de bord (dashboard). Les noyaux sont des petits programmes qui traitent des requêtes dans un langage particulier et qui réagissent avec les réponses correspondantes. Le noyau standard est l'IPython, un interprétateur de lignes de commande, qui permet de travailler avec Python. On trouve en plus une bonne cinquantaine d'autres noyaux qui permettent l'utilisation d'autres

langages, comme C++, R, Julia, Ruby, JavaScript, CoffeeScript, PHP ou Java. Le tableau de bord sert d'une part d'interface de gestion des différents noyaux, et d'autre part de centrale pour la création de nouveaux documents Notebook, ou pour ouvrir des documents existants. Jupyter Notebook est disponible sous licence BSD modifiée et donc librement utilisable par tous.

Tensorflow / keras

Créé par l'équipe Google Brain en 2011, sous la forme d'un système propriétaire dédié aux réseaux de neurones, TensorFlow s'appelait à l'origine DistBelief. Par la suite, le code source de DistBelief a été modifié et cet outil est devenu une bibliothèque basée application. En 2015, il a été renommé TensorFlow et Google l'a rendu open source. Depuis lors, il a subi plus de 21000 modifications par la communication et est passé en version 1.0 en février 2017.

TensorFlow est une bibliothèque de Machine Learning, une boîte à outils permettant de résoudre des problèmes mathématiques complexes. Celle-ci regroupe un grand nombre de modèles et d'algorithmes de Machine Learning et de Deep Learning. Son API front-end de développement d'applications repose sur le langage de programmation Python, tandis que l'exécution de ces applications s'effectue en C++ haute-performance.

Keras est une API de réseaux de neurones de haut niveau, écrite en Python et interfaçable avec TensorFlow, CNTK et Theano. Elle a été développée avec pour objectif de permettre des expérimentations rapides. Être capable d'aller de l'idée au résultat avec le plus faible délai possible étant la clef d'une recherche efficace.

Keras permet le prototypage rapide et facile (de par sa convivialité, sa modularité et son extensibilité), supporte à la fois les réseaux convolutifs et les réseaux récurrents ainsi que la combinaison des deux, et fonctionne de façon transparente sur CPU et GPU.

Jeux de données

Utilisation des auto-encodeurs sur des données véhiculaires.

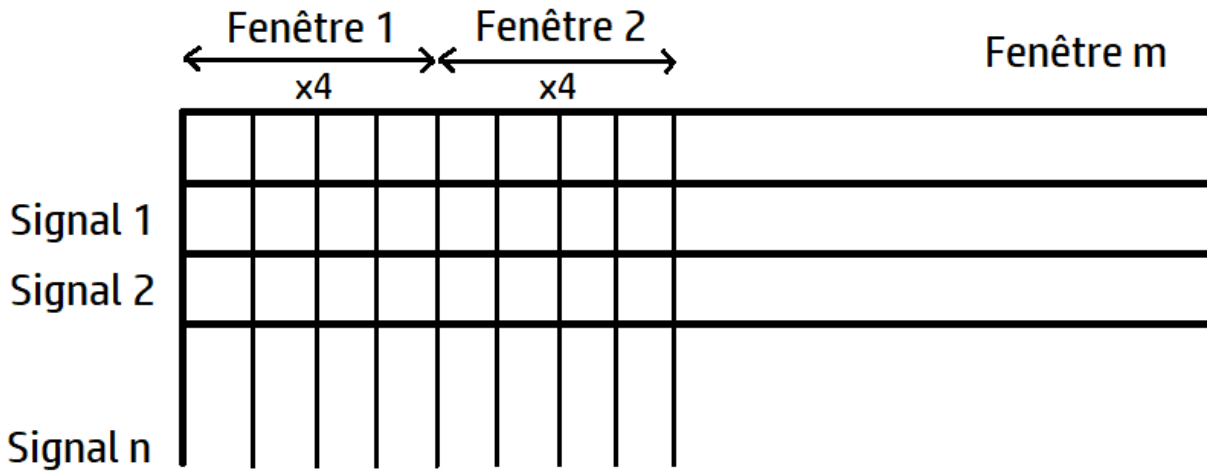
Il s'agit de deux matrices : Quantized_100.csv et Sampled_100.csv représentant 2282 signaux.

Initialement, ces signaux représentent des valeurs de 4 variables :

- Cap (heading) ;
- Vitesse (speed) ;
- Accélération longitudinale (longitudinalacceleration) ;
- Taux de lacet (yawrate).

Elles sont observées toutes les secondes pendant des trajets de durées différentes.

Chacun des 2282 signaux est divisé en 100 fenêtres de temps dont la taille et proportionnelle à la longueur du signal.



Chaque fenêtre contient 4 valeurs, une pour chaque variable. Chaque fenêtre dure un certain nombre de secondes durant lesquelles des mesures sont prises pour chaque variable.

Par exemple, dans une fenêtre de 6 secondes, nous récupérons 6 mesures pour chaque variable observée.

Dans le cas de la matrice quantifiée Quantized, pour chaque variable, chaque cellule correspond à la moyenne de ces mesures dans la fenêtre correspondante.

Dans le cas de la matrice quantifiée Quantized, pour chaque variable, chaque cellule correspond à une valeur prise parmi ces mesures dans la fenêtre correspondante.

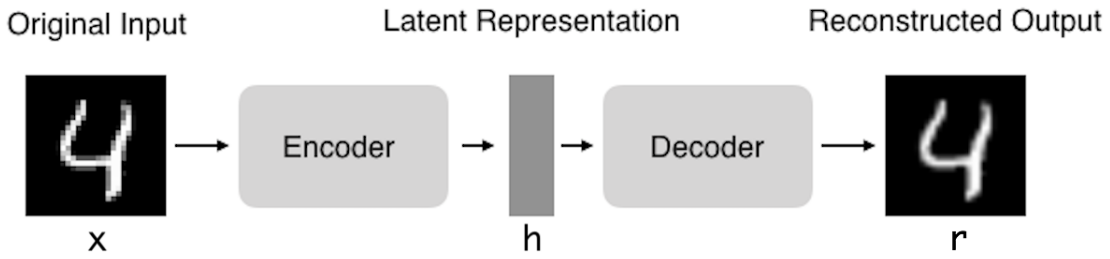
Par la suite, il a fallu former des sous-matrices par variable. Ainsi, comme il y a 4 variables, il suffisait d'extraire une colonne toutes les 4 colonnes.

Par exemple, la première variable observée étant heading, il a fallu extraire les colonnes 1, 5, 9, 13, etc. pour former la première sous-matrice. Puis, il a fallu se déplacer d'un indice pour la seconde variable qui correspond aux colonnes 2, 6, 10, 14, etc., idem pour la troisième (colonnes 3, 7, 11, 15, etc.), puis la quatrième et dernière (colonnes 4, 8, 12, 16, etc.).

Auto-encodeur

Les auto-encodeurs sont un type spécifique de réseaux de neurones à propagation avant (feedforward) utilisés pour l'apprentissage non-supervisé de caractéristiques discriminantes. Ils compriment l'entrée en un « code » de dimension réduite et reconstruisent ensuite la sortie à partir de cette représentation. Le code est un « résumé » compact ou une « compression » de l'entrée, également appelée représentation de l'espace latent.

Un auto-encodeur se compose de 3 éléments : l'encodeur, le code et le décodeur. La couche cachée compresse l'entrée et produit le code, le décodeur reconstruit ensuite l'entrée uniquement à l'aide de ce code.



Pour construire un auto-encodeur, nous avons besoin de 3 choses :

1. Une méthode d'encodage ;
2. Une méthode de décodage ;
3. Une fonction de coût pour comparer la sortie avec la cible.

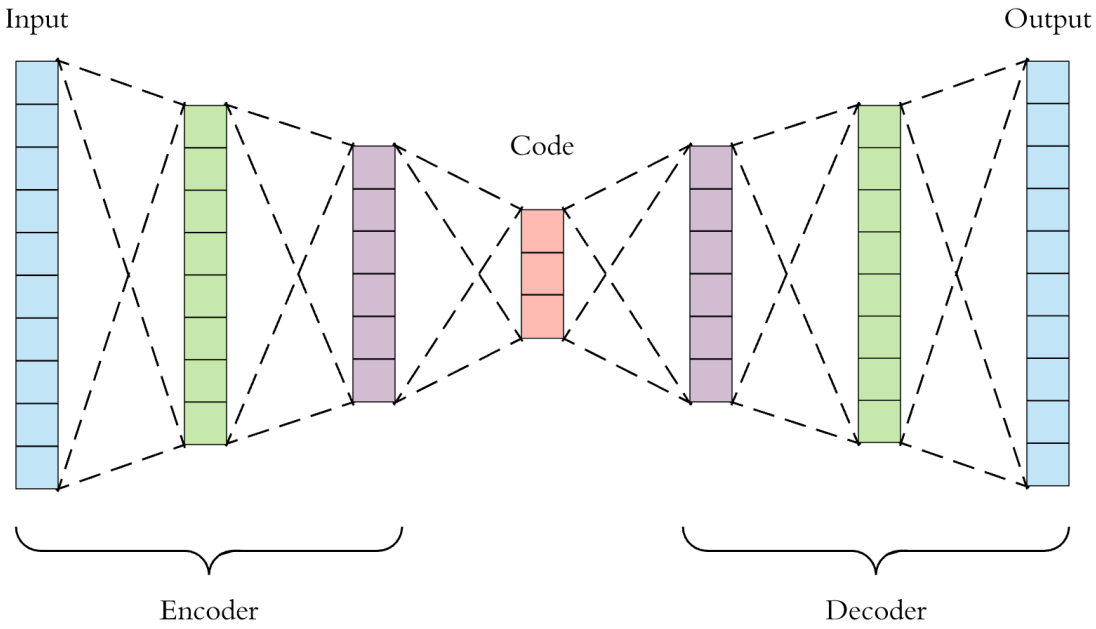
Les auto-encodeurs sont principalement des algorithmes de réduction de dimensionnalité (ou de compression) avec quelques propriétés importantes. En effet, ils sont :

- Spécifique aux données : Les auto-encodeurs ne sont capables de compresser de que des données similaires à celles sur lesquelles ils ont été formés. Comme ils apprennent des caractéristiques spécifiques aux données d'entraînement, ils sont différents d'un algorithme de compression standard tel que gzip. On ne peut donc pas s'attendre à ce qu'un auto-encodeur entraîné sur des chiffres manuscrits compresse des photos de paysages ;
- Avec perte : La sortie de l'auto-encodeur ne sera pas exactement la même que l'entrée, il s'agira d'une représentation proche mais dégradée ;
- Non-supervisés : Pour entraîner un auto-encodeur, des données brutes suffisent. Les auto-encodeurs sont considérés comme une technique d'apprentissage non-supervisée car il n'est pas nécessaire de nourrir le réseau avec des données étiquetées. Mais pour être plus précis, ils sont auto-supervisés car ils génèrent leurs propres étiquettes à partir des données d'entraînement.

Les auto-encodeurs sont étroitement liés à l'analyse en composantes principales (ACP). En fait, si la fonction d'activation utilisée dans l'auto-encodeur est linéaire dans chaque couche, les variables latentes présentes au niveau du goulot d'étranglement (la plus petite couche du réseau, alias « code ») correspondent directement aux composantes principales de l'ACP. Dans le cas contraire, il peut proposer différents niveaux d'abstraction et est capable de restituer des « patterns » non-linéaires. En général, la fonction d'activation utilisée dans les auto-encodeurs est non-linéaire, les fonctions d'activation les plus utilisées sont les fonctions ReLU (Rectified Linear Unit) et sigmoïde.

Architecture

Comme susmentionné, l'auto-encodeur est composé d'un encodeur et d'un décodeur. Ils sont tous deux des réseaux neuronaux entièrement connectés, l'auto-encodeur dans sa forme la plus basique est donc composés d'ANN, mais il est possible d'utiliser des CNN ou bien des GAN par exemple. Le code quant à lui, est une couche cachée avec une dimension choisie arbitrairement, la taille du code est donc un hyperparamètre.



Ainsi, l'encodeur est le modèle de reconnaissance qui encode les données sous forme d'une représentation compressée (code) dans un espace de dimension réduit (espace latent). L'image compressée apparaît comme une déformation de l'image originale. D'autre part le décodeur est le modèle génératif qui décode la représentation compressée (le code) à sa dimension originale. L'image décodée est une reconstruction approximative de l'image originale.

L’architecture de l’auto-encodeur est généralement symétrique, autrement dit le décodeur correspond à l’architecture inverse de l’encodeur, mais ce n’est pas toujours le cas. En revanche, pour les raisons précédemment évoquées, les dimensions de l'entrée et de la sortie doivent être les mêmes.

Il y a 4 hyperparamètres à définir avant d’entraîner un auto-encodeur :

1. La taille du code : Nombre de nœuds dans la couche centrale. Plus la taille est petite, plus le taux de compression est important ;
2. Le nombre de couches : L'auto-encodeur peut être aussi profond que souhaité ;
3. Le nombre de nœuds par couche : Le nombre de nœuds par couche diminue avec chaque couche suivante de l'encodeur, et augmente à nouveau dans le décodeur ;
4. La fonction de coût : nous utilisons soit la méthode des moindres carrés, soit l’entropie croisée binaire. Si les valeurs d'entrée sont dans la plage $[0, 1]$, nous utilisons généralement la seconde, sinon nous utilisons la première.

Les auto-encodeurs sont entraînés de la même manière que les ANN, par rétropropagation du gradient.

Comme l'entrée et la sortie son approximativement les mêmes, il ne s'agit pas vraiment d'un apprentissage non-supervisé en réalité, c'est pourquoi nous appelons généralement cela un apprentissage auto-supervisé. Le but de l'auto-encodeur est de sélectionner nos fonctions d'encodage et de décodage de telle sorte que nous ayons besoin du minimum d'informations pour encoder les données, puis les reconstruire de l’autre côté.

Si nous utilisons trop peu de nœuds dans la couche centrale, la reconstruction des données sera limitée et celles-ci seront bruitées ou trop dégradées par rapport à l'entrée. Si nous utilisons trop de nœuds, il n'y a aucun intérêt à utiliser la compression.

Le cas de la compression est assez simple, chaque fois que vous téléchargez quelque chose sur Netflix, par exemple, les données qui vous sont envoyées sont compressées. Une fois qu'elles arrivent sur votre ordinateur, elles passent par un algorithme de décompression et vous sont ensuite affichées. C'est analogue à la façon dont fonctionnent les fichiers zip, mais cela se fait en coulisses via un algorithme de streaming.

Implémentation

Tout d'abord, le jeu de données utilisé a été divisé en 2 parties X_train et X_test suivant un ratio 80/20.

```
import numpy as np

#split into train and test sets
X_quantized = np.loadtxt('datasets/Quantized_100.csv')
X_quantized_20 = np.loadtxt('datasets/Quantized_20.csv')
X_sampled = np.loadtxt('datasets/Sampled_100.csv')
X_sampled_20 = np.loadtxt('datasets/Sampled_20.csv')

print(X_quantized.shape, X_quantized_20.shape, X_sampled.shape, X_sampled_20.shape)
```

```
(2282, 400) (2282, 80) (2282, 400) (2282, 80)
```

```
train_size = int(len(X_quantized) * 0.80)
test_size = len(X_quantized)-train_size
X_train, X_test = X_quantized[0:train_size, :], X_quantized[train_size:len(X_quantized), :]
```

À noter que chacun de ces jeux de données a également été divisé en sous-matrices (une par variable) en vue de réaliser des tests dessus et mettre en évidence les variables les plus impactantes.

Ensuite, les données ont été normalisées :

```
from sklearn import preprocessing
# normalize the data attributes
X_train = preprocessing.normalize(X_train)
X_test = preprocessing.normalize(X_test)
```

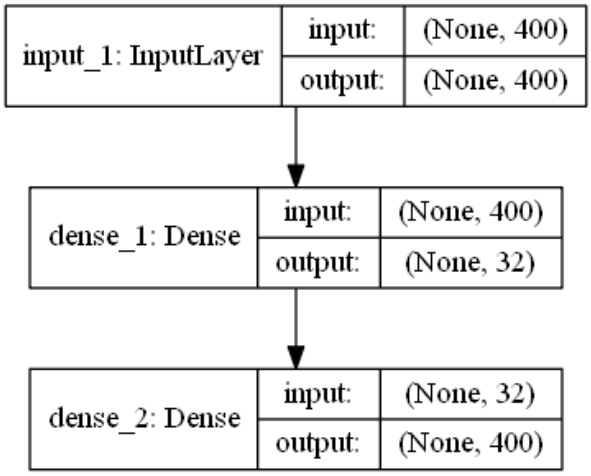
Auto-encodeur ANN

L'auto-encodeur a pu être conçu à partir d'une classe préalablement créée :

```
(encoder, decoder, autoencoder) = ANN_autoencoder.build(400)
autoencoder.compile(optimizer = 'adam', loss = 'mse')
```

La méthode de descente de gradient utilisée est Adam et la fonction de coût une méthode des moindres carrés.

Architecture de l'auto-encodeur :

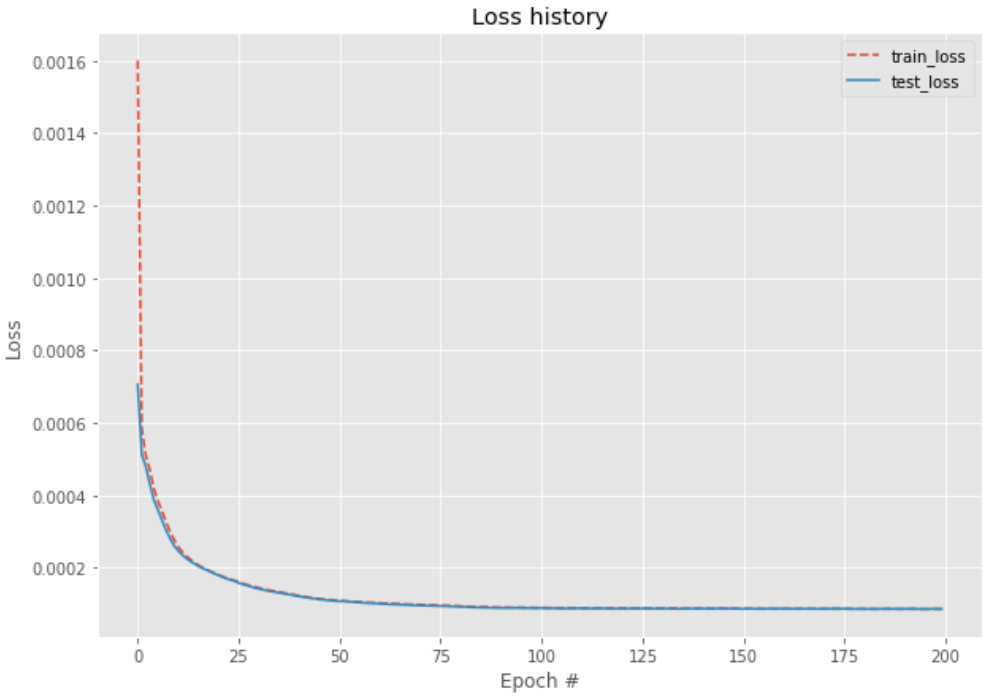


L'architecture utilisées est basique et se compose uniquement de couches entièrement connectées. Le taux de compression est de 12.5, le code étant composé de 32 neurones.

Le modèle fut entraîné sur 200 époques, une taille de batch de 64 et des données mélangées à chaque passe.

```
H = autoencoder.fit(X_train, X_train, epochs = 200,
                    batch_size = 64, shuffle = True,
                    verbose = 1, validation_data = (X_test, X_test))
```

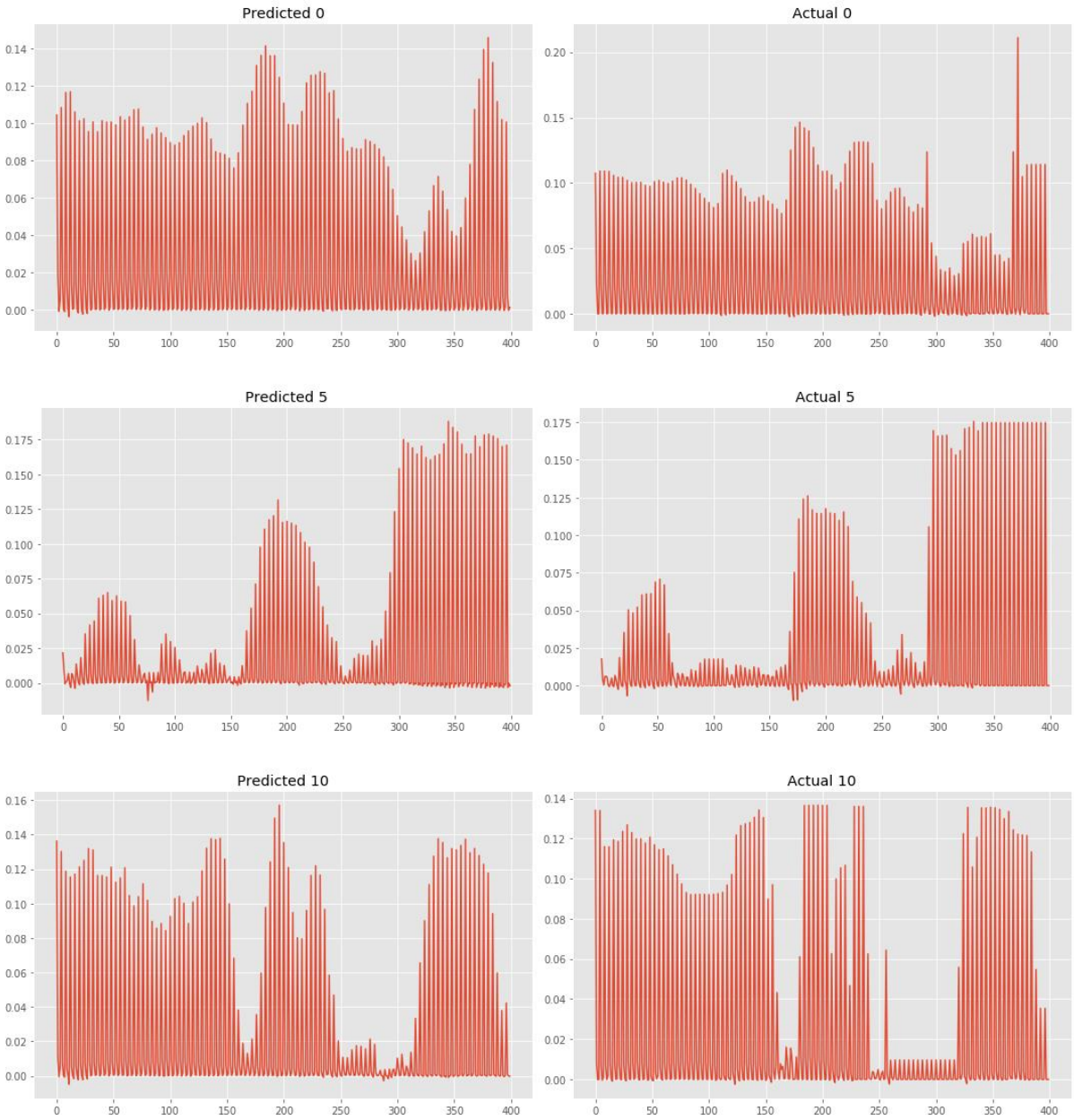
Visualisation :

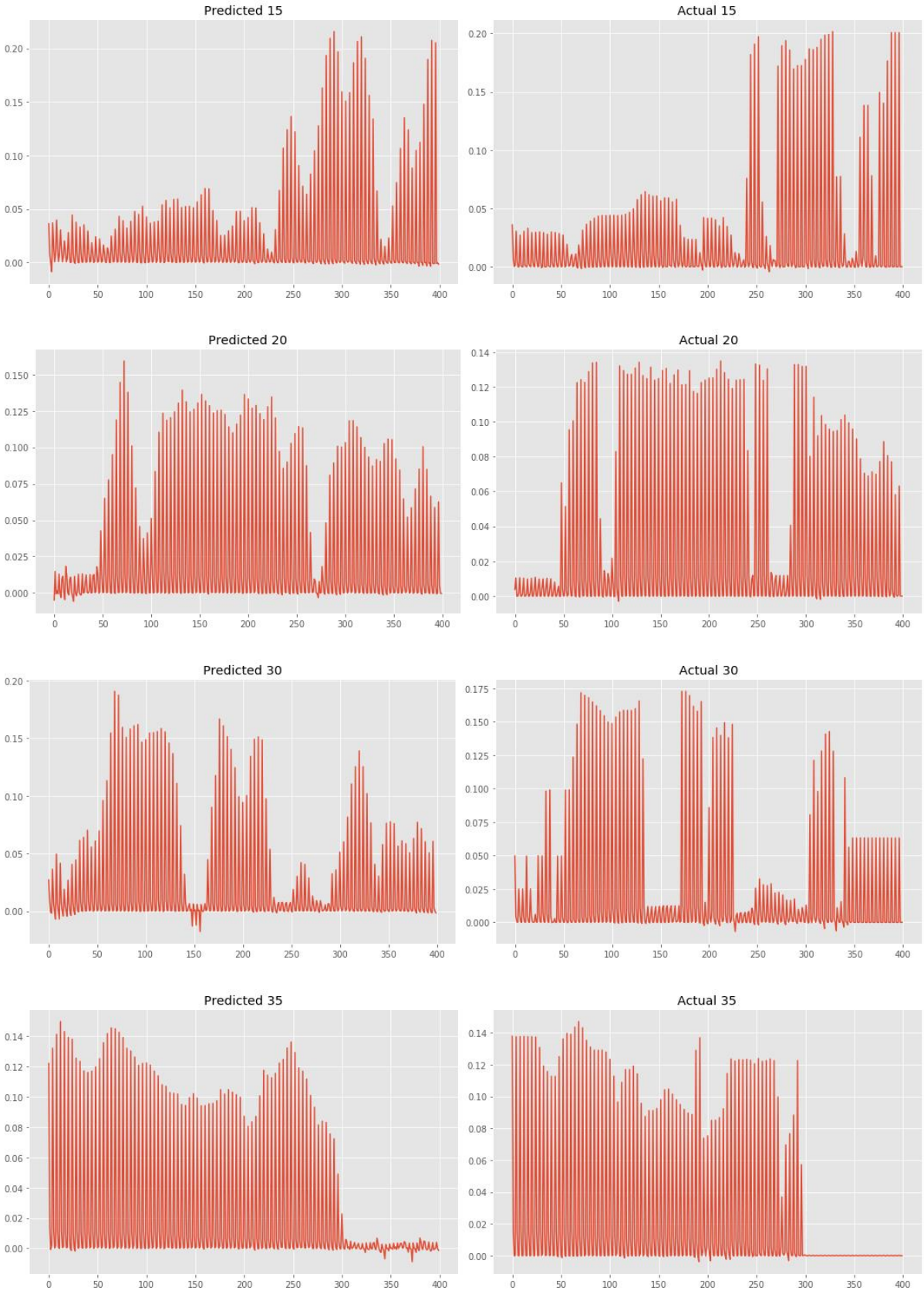


Prédictions :

```
decoded_imgs = autoencoder.predict(X_test)

plt.style.use("ggplot")

for i in range(0, 51, 5):

    fig, ax = plt.subplots(1, 2, figsize = (15, 5))

    ax[0].set_title('Predicted ' + str(i))
    ax[0].plot(decoded_imgs[i, :])
    ax[1].set_title('Actual ' + str(i))
    ax[1].plot(X_test[i, :])

    fig.tight_layout()
```

À gauche le signal reconstitué, à droite le signal d'origine.

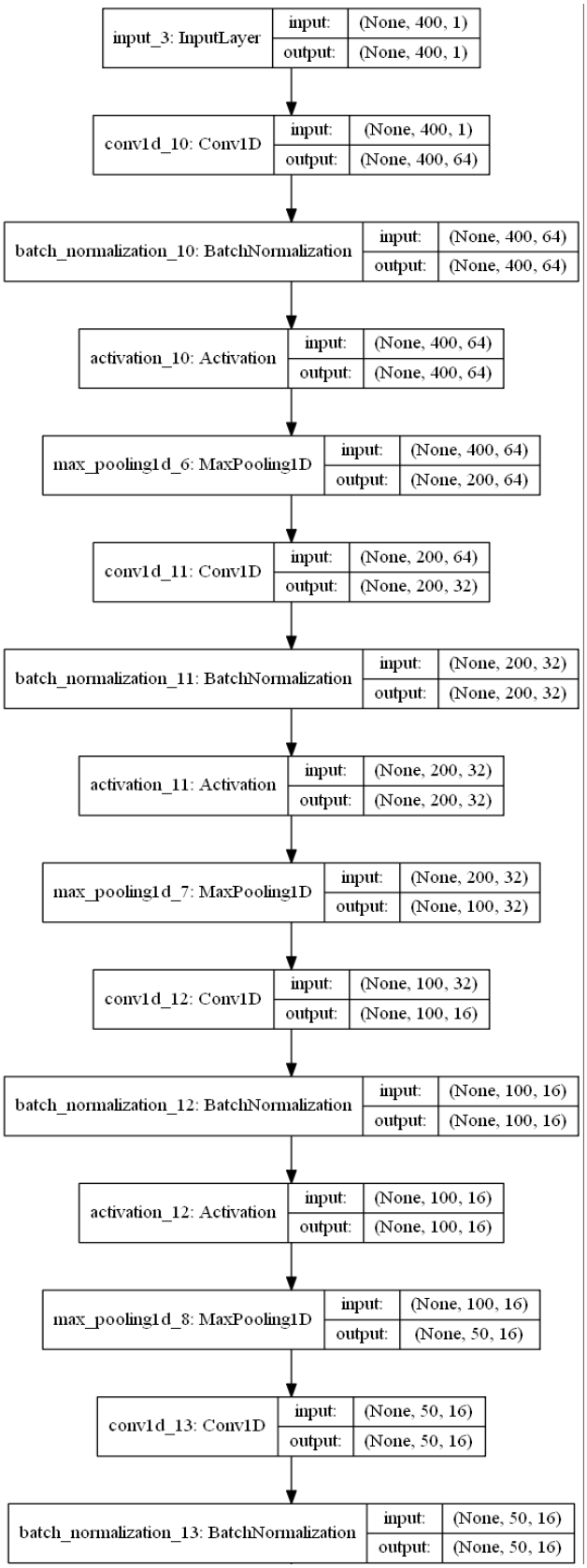


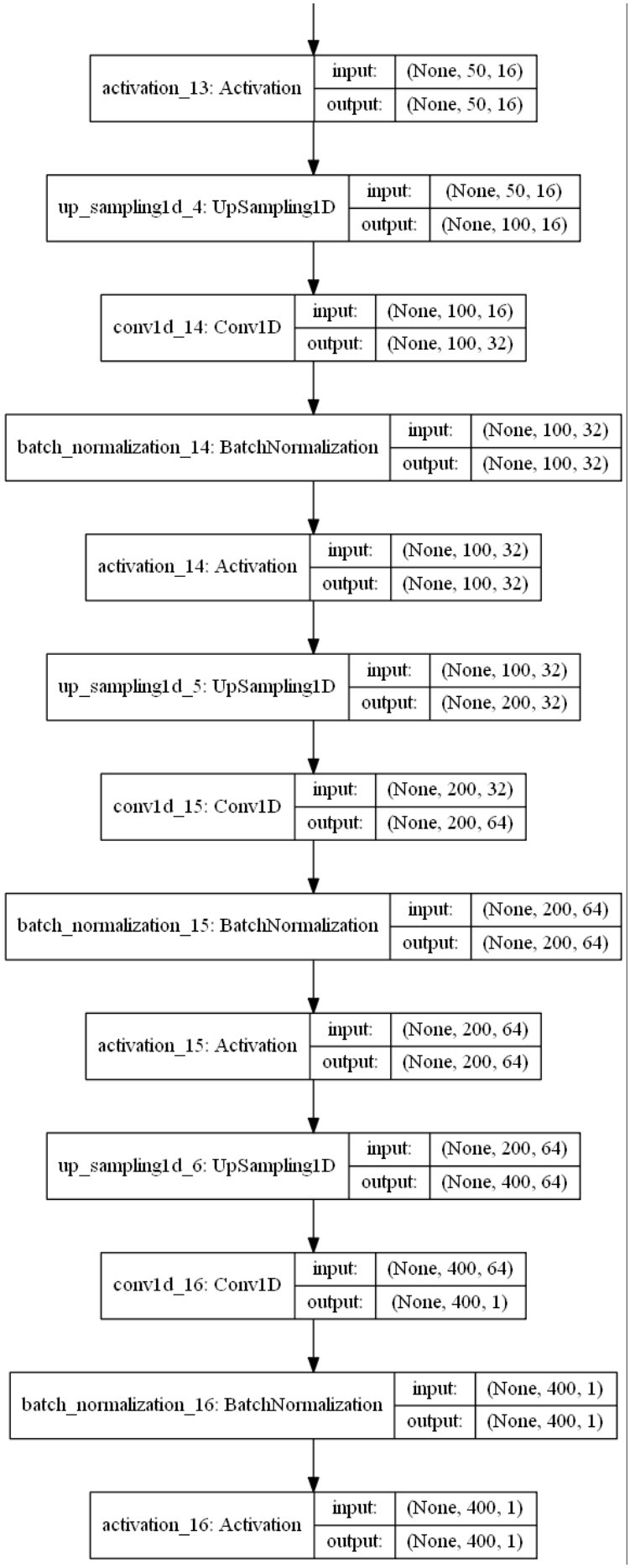


Nous observons des résultats plutôt satisfaisants, il y a évidemment une perte compte tenu de la méthode (auto-encodeur) utilisée mais la reconstitution de signal est semblable au signal d'origine.

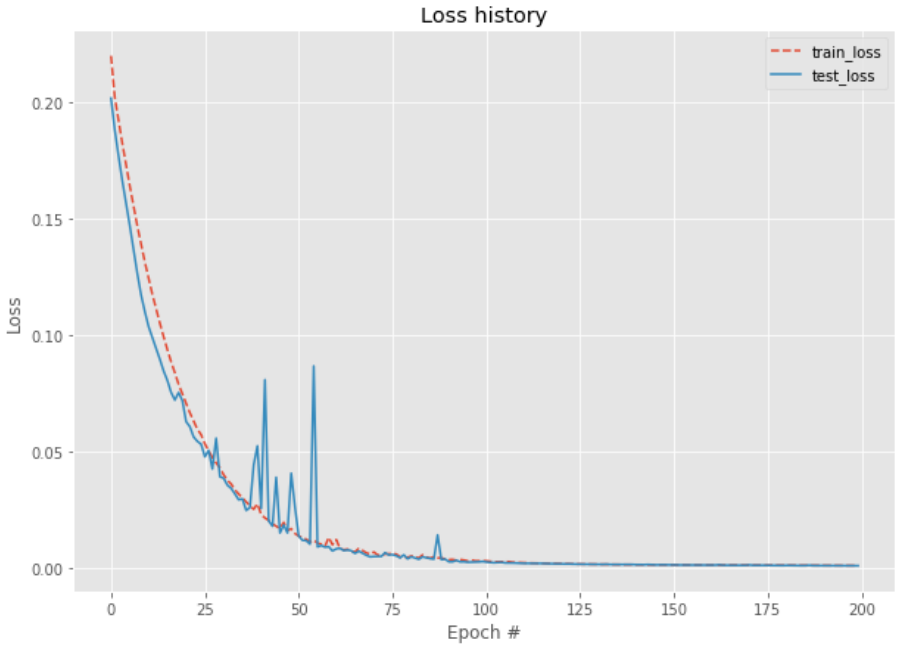
Auto-encodeur CNN

Un second auto-encodeur a été créé, cette fois-ci à partir d'une architecture à convolution.

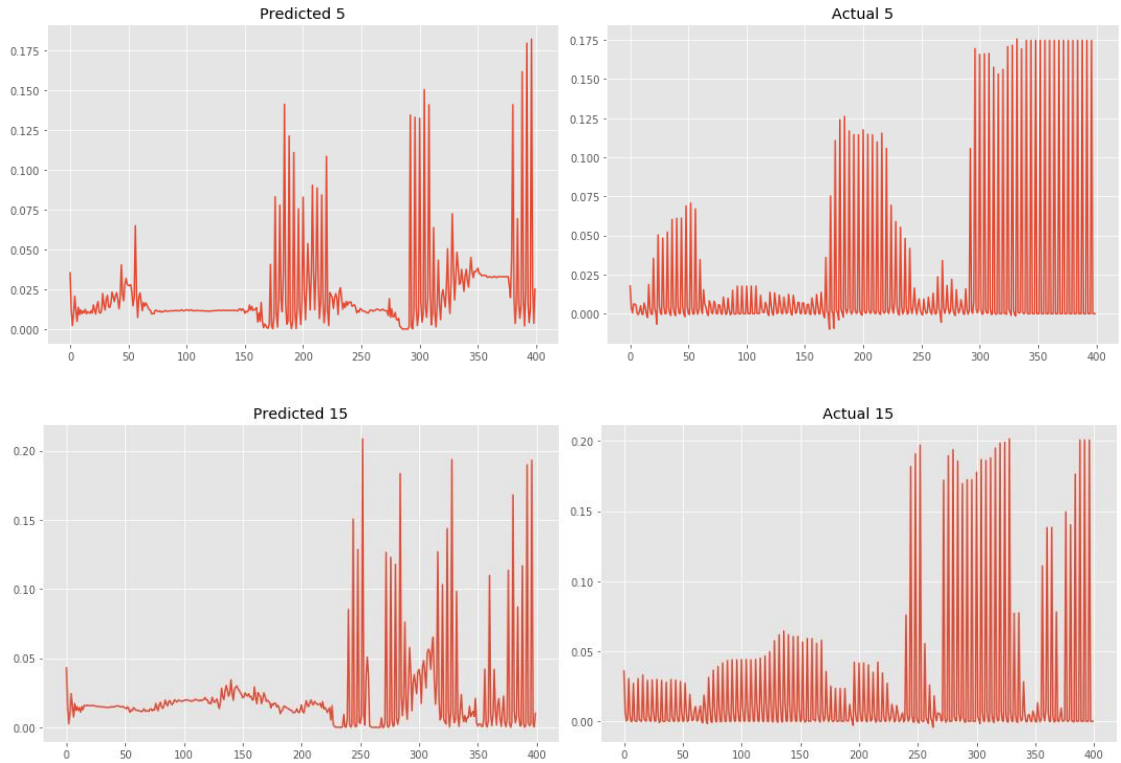



activation_13: Activation
input: (None, 50, 16)
output: (None, 50, 16)
up_sampling1d_4: UpSampling1D
input: (None, 50, 16)
output: (None, 100, 16)
conv1d_14: Conv1D
input: (None, 100, 16)
output: (None, 100, 32)
batch_normalization_14: BatchNormalization
input: (None, 100, 32)
output: (None, 100, 32)
activation_14: Activation
input: (None, 100, 32)
output: (None, 100, 32)
up_sampling1d_5: UpSampling1D
input: (None, 100, 32)
output: (None, 200, 32)
conv1d_15: Conv1D
input: (None, 200, 32)
output: (None, 200, 64)
batch_normalization_15: BatchNormalization
input: (None, 200, 64)
output: (None, 200, 64)
activation_15: Activation
input: (None, 200, 64)
output: (None, 200, 64)
up_sampling1d_6: UpSampling1D
input: (None, 200, 64)
output: (None, 400, 64)
conv1d_16: Conv1D
input: (None, 400, 64)
output: (None, 400, 1)
batch_normalization_16: BatchNormalization
input: (None, 400, 1)
output: (None, 400, 1)
activation_16: Activation
input: (None, 400, 1)
output: (None, 400, 1)

Hormis l'architecture, le code produit pour concevoir cet auto-encodeur est similaire au précédent, à la différence près que l'algorithme Adadelta a été utilisé à la place de Adam.



Prédictions :



Dans le cas présent, la forme du signal est comparable au signal original mais dans l'ensemble le résultat obtenu n'est pas satisfaisant.

Difficultés rencontrées

Compte tenu de la situation sanitaire du pays durant les mois de mars, avril et mai, les changements de modalités d'évaluation du Master ne m'ont laissé que peu de temps supplémentaire afin de travailler sur le projet TER. En effet, les examens écrits ayant été annulés, ceux-ci sont devenus des projets qui qui furent à réaliser dans des conditions particulières et en parallèle d'un stage de 35h/semaine en télétravail.

L'auto-encodeur est une méthode particulière d'apprentissage non-supervisé, il aurait été intéressant de parvenir à des résultats satisfaisants avec l'utilisation de CNN, voire de GAN (réseaux antagonistes génératifs), de tester davantage d'architectures ou même de variantes telles que l'auto-encodeur variationnel. De plus, cela aurait permis de faire du clustering sur les données véhiculaires fournies.

Bilan du projet

Ce projet a été réalisé dans le cadre du module d'INFO0809. Celui-ci convoque des connaissances variées et fait le lien entre plusieurs modules de notre formation, afin de concevoir plusieurs réseaux de neurones dans le cadre d'un apprentissage non-supervisé. Il a été l'occasion pour moi, étudiant, de consolider des compétences, puis en acquérir de nouvelles en implémentant plusieurs réseaux de neurones basés sur une approche nouvelle et une architecture singulière.