

Résolution du problème du sac a dos multi-objectifs avec le Deep Learning.





Table des matières

[1. Introduction](#_cjo3vltqnumk) **3**

[2. Problème du sac a dos mono-objectif avec le Deep Learning](#_9uiqkwkggzf4) **4**

[2.1 Genèse des données](#_yhaxuixvzhe8) **6**

[2.2 Transformation des données](#_940yzth0m8yp) **9**

[2.3 Modélisation](#_cdxuzczdsjo3) **12**

[3. Problème du sac a dos multi-objectif avec le Deep Learning](#_u7n1l8mor4g3) **15**

[3.1 Genèse des données](#_axomoigl882q) **16**

[3.2 Modélisation](#_ir99p0x1vdpp) **22**

[4. Conclusion](#_9qb0cedffj5t) **23**

# 1. Introduction

L’optimisation combinatoire est une branche de la programmation mathématique exploitée dans plusieurs domaines notamment la théorie des enchères, le génie logiciel et l’informatique théorique.

L’optimisation combinatoire consiste à trouver la meilleure solution parmi un nombre fini de choix, et qui recouvre les méthodes qui servent à déterminer l’optimum d’une fonction sous des contraintes données.

Parmi les problèmes typiques de cette branche il y a le problème du commerçant voyageur (Travelling Salesman Problem TSP) et le problème du sac à dos (Knapsack Problem) qui peuvent être résolus grâce à des algorithmes de recherches ou des métaheuristiques.

Dans ce projet, on s'intéresse particulièrement à la version multi-objectifs du problème du sac a dos (Multi-objective Knapsack Problem MKP), une explication détaillée du problème est disponible sur ce lien: [Problème du sac à dos — Wikipédia](https://fr.wikipedia.org/wiki/Probl%C3%A8me_du_sac_%C3%A0_dos#Sac_%C3%A0_dos_multi-objectif).

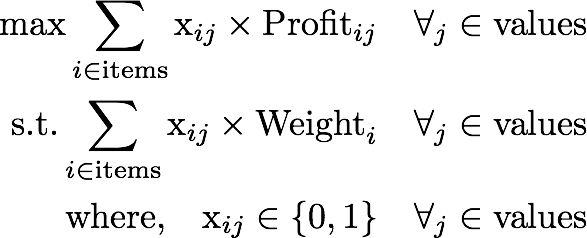
L’objectif est de construire un modèle prédictif de réseau de neurones capable de trouver la solution qui maximise les fonctions objectifs tout en tenant compte des différentes contraintes et des capacités des sac a dos.

# 2. Problème du sac a dos mono-objectif avec le Deep Learning

# 

Dans cette partie, et dans le but de simplifier la démarche, nous allons traiter le cas le plus simple du problème qui consiste à choisir d’un ensemble d’objet, un sous ensemble ou la somme des valeurs des objets choisis (fonction objectif) est maximale sans que la somme de leurs poids dépasse la capacité du sac à dos (contrainte).

Modéle mathématique du probleme:



Pour ce faire, nous allons entraîner un réseau de neurones sur des instances de Benchmark où les solutions sont connues au préalable, pour que notre modèle soit capable de prévoir des solutions optimales des futurs instances.

Une Instance de Benchmark présente un ensemble d’objets, leurs valeurs et poids, la capacité du sac à dos et la solution optimale.

En d’autre termes, à chaque fois que notre modèle envisage un ensemble d’objets et un sac à dos , il doit décider de ceux à retenir et ceux à rejeter de sorte qu’on obtient le profit le plus élevé sans violation de la contrainte.

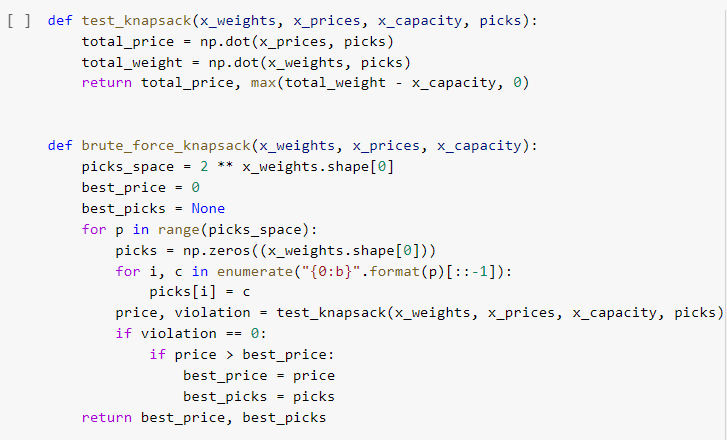
Les étapes par lesquelles nous allons passer sont:

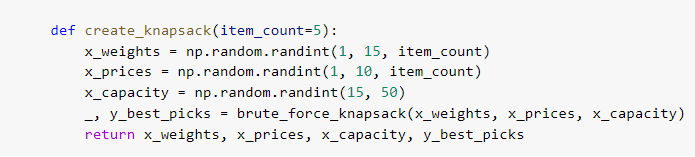
* Génération des données (instances d'entraînement) selon une logique aléatoire.
* Transformation des données au format adéquat.
* Construction du modèle.
* Entraînement et évaluation.

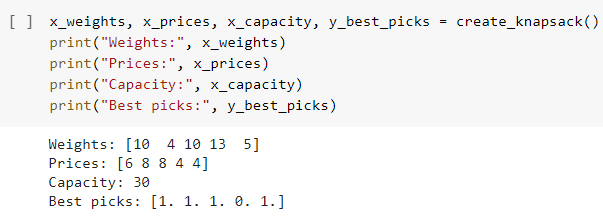
# 2.1 Genèse des données

Les captures ci-dessous correspondent au code en python de la partie de génération de données disponible sur [REF1](https://github.com/shshemi/NeuralKnapsack), et la description de chaque fonction:

* Create\_knapsack\_dataset:   
  - permet de générer le nombre désiré d’instances.  
  - retourne un tableau X contenant les poids et les valeurs des objets et la capacité du sac à dos, et un tableau Y contenant les solutions optimales correspondante à chaque instance.
* Create\_knapsack:   
  permet de générer une seule instance et sa solution.
* Brute\_force\_knap\_sack:  
  retourne la solution optimale a partir d’une instance données.
* Test\_knap\_sack:  
  Tester si la combinaison choisie a l’itération courante respecte la contraintes ou pas.







# 

# 

# 

# 

# 

# 

# 

# 

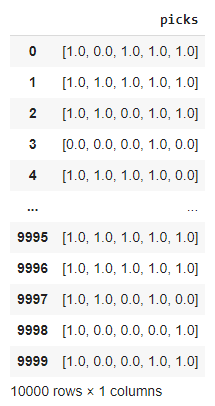
# 

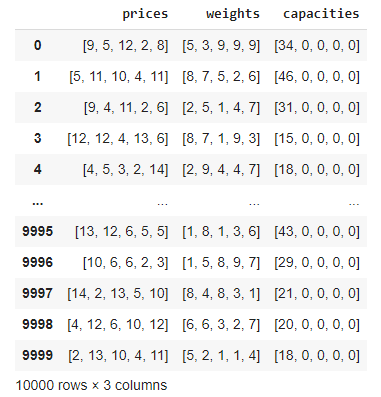
# 

# 2.2 Transformation des données

Comme déjà mentionné dans la partie descriptive du code de la partie précédente, la sortie est sous forme de deux tableau, l’un contient les informations relatives à chaques instance et l’autre contient respectivement les solutions correspondantes, mais avant de passer à la phase de modélisation, et pour que nos données soient compréhensible par le modéle sans perdre de sens, il faut extraire chaque instance à part sous forme d’une matrice contenant toutes les informations relative à cette dernière, et pour ce faire on exécute le code suivant:



On obtient ainsi deux “ Pandas DataFrame” qui encapsulent nos données d'entraînement de la forme suivante :



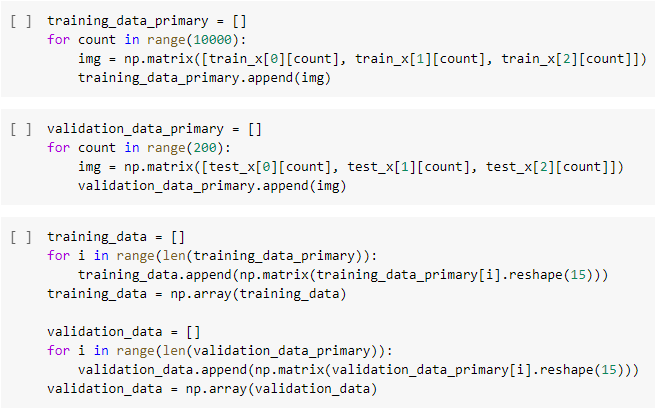
D’après cette figure, nous disposons d’instances de 5 objets, et

à partir de ce résultat, nous allons créer les matrices de sorte que chacune ait la forme suivante:



Ou V1: valeur du premier élément de l’instance, P1: poids du premier élément de l’instance et C: capacité du sac à dos.

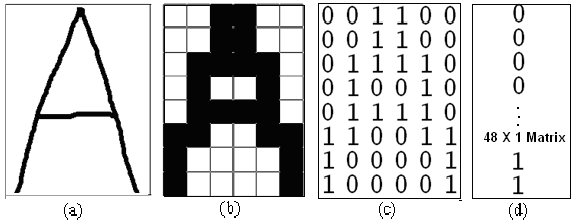
Le code ci-dessous renvoie la forme finale des données d'entraînement (10000 matrice), et de test (200 matrice)



Une autre raison pour laquelle cette transformation est primordiale, c’est que les modèles de réseau de neurones ne supportent que les entrées convertibles en “Tenseur” (format ou type de données).

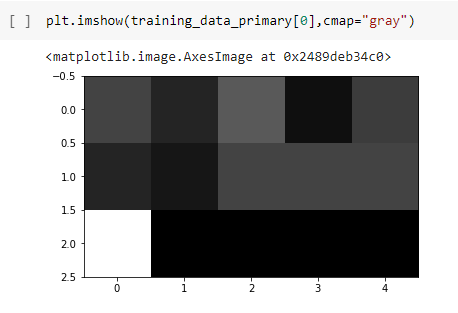
# 2.3 Modélisation

Il n’est peut être pas encore très convaincant qu’il y ait un lien entre l’optimisation combinatoire et le Deep Learning, mais après avoir transformé les données d’entrée en matrices, et en faisant l’analogie avec le “Computer Vision” -qui représente la manière dont les ordinateurs peuvent acquérir une compréhension de haut niveau à partir d’images numériques- on devient plus convaincu de ce lien car les images sont traités comme des matrices, ou on associe a chaque pixelle de l’image un nombre allant de 0 à 255 représentant la densité de la couleur de cette pixelle dans un certain canal (gris, rouge, vert ou bleu).

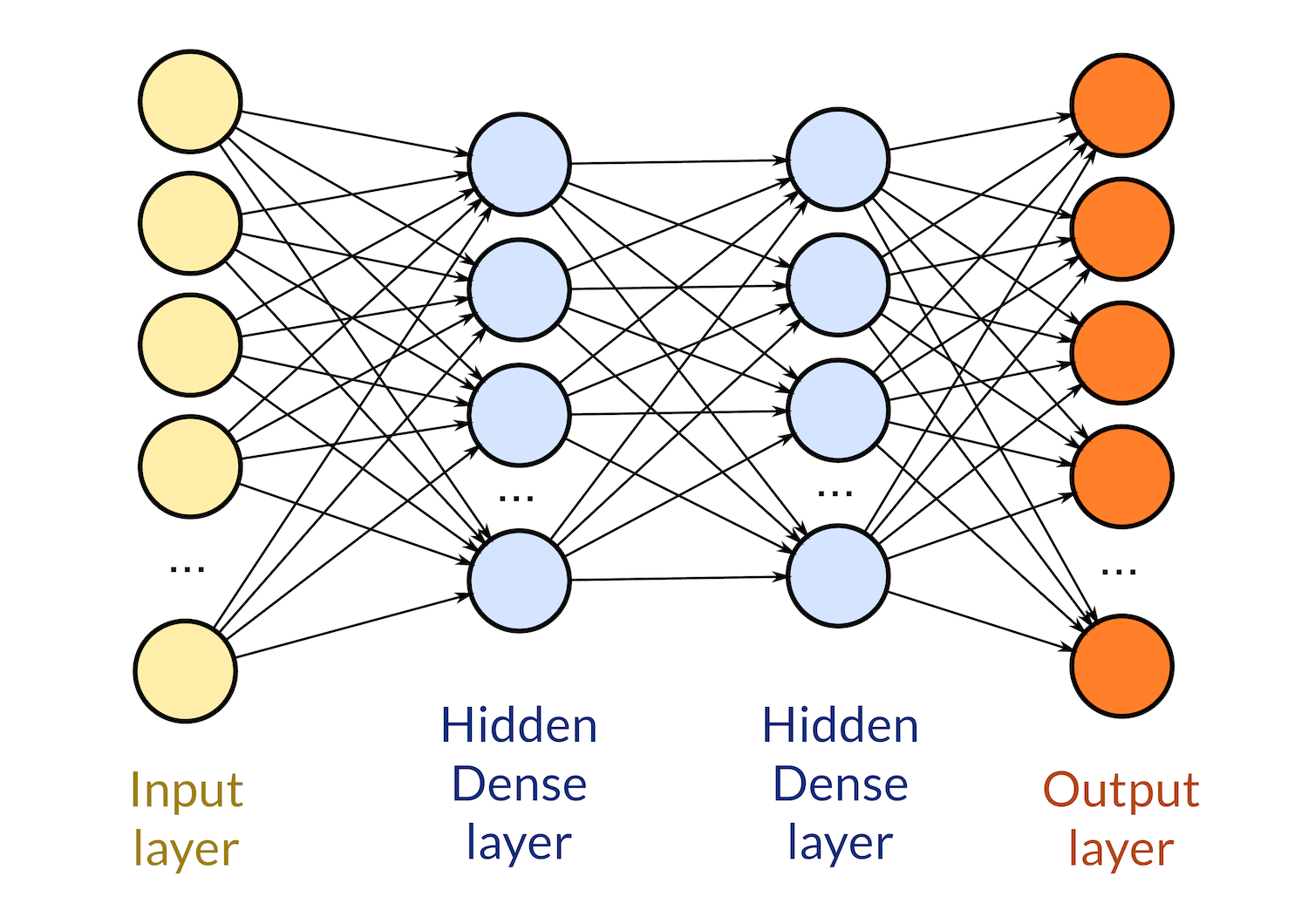


Dans notre cas, c’est comme si on disposait de 10000 images et pour chaque image on associe un tableau de variables binaires qui est à la base la solution de l’instance relative à l'image en question.

Si on affiche la première matrice (instance) de nos données d'entraînement comme étant une matrice correspondante a une image on obtient la figure suivante:

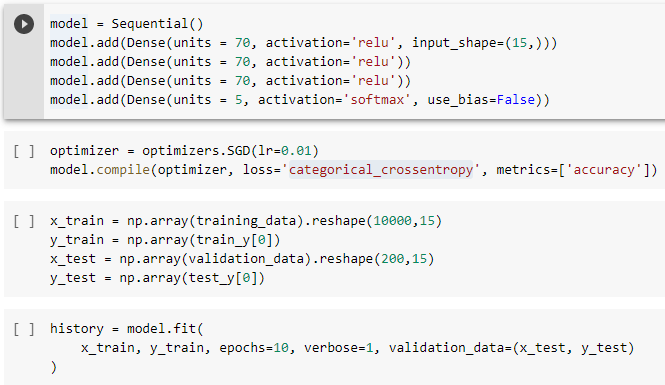


Une fois arrivé à cette étape, il ne reste qu'à construire le modèle, et pour ce projet, nous nous sommes contentés d’une architecture très simple, une couche d’entrée, deux couches cachés et une couche de sortie comportant 5 neurones ou chacune contiendra la valeur binaire qui représente la décision du modèle par rapport à chaque objet comme illustré ci-dessous:



* La méthode d’optimisation de calcul de perte (Optimizer) choisie et la descente de gradient stochastique (Stochastic Gradient Descent SGD).
* La fonction de calcul perte (Loss Function) convenable est la   
  “Categorical Cross Entropy” vu qu’on est dans le cas où nos sorties sont des variables catégoriques.
* La fonction d’activation utilisée dans les couches cachées est la “Rectified Linear Unit ReLU”.
* La fonction d’activation utilisée dans la couche de sortie est la “Softmax” vu que notre modèle est à sortie multiple.
* On s’est limité à la précision comme mesure d’évaluation.

Ci-dessous le code relatif à la modélisation:



On obtient avec ce modèle qui n’est pas encore optimisé et loin d'être à l'état de l’art, une précision de ~80%.

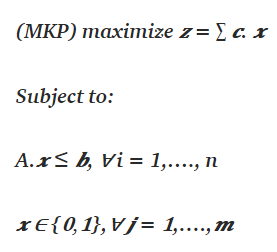
# 3. Problème du sac a dos multi-objectif avec le Deep Learning

Après avoir traité le cas d’une seule fonction objectif, le principe et la démarche deviennent plus claires, l’idée c’est d’encapsuler les informations relatives a chaque problème ou instance de benchmark dans une matrice, et associer a chaque matrice un tableau de variables binaires (solution) ou chacune représente la décision de rejet ou rétention d’un objet.

Nous allons suivre la même logique pour traiter le problème a 10 fonctions objectifs et 10 contraintes. Autrement dit, c’est le cas où on dispose de 10 sacs à dos et chacun a une capacité différente.

La solution optimale relative à une instance est celle qui maximise le profit des 10 sacs combinés.

Modèle mathématique:



Ou **c** le vecteur des profits par objet, **b** le vecteur des capacités de chaque sac a dos, **x** le vecteur des variables binaires et **A** est la matrice des coefficients des contraintes.

# 3.1 Genèse des données

La finalité de ce travail **(3)** est aussi d’avoir un modèle de réseau de neurones dont le rôle est de renvoyer une solution de sorte que les valeurs des fonctions objectifs soient le plus élevées tout en respectant les contraintes. La différence entre ce problème et celui qui précède ne persiste que du point de vue technique: la manière de collecte et restructuration des données d'entraînement et de test. Cette fois-ci nous ne pouvons pas tout simplement générer les données suivant une logique aléatoire et puis essayer toutes les combinaisons possibles et comparer a chaque fois le profit pour déduire la solution optimale, nous allons utiliser des instances de benchmark recueillis de vrais problèmes écrits sur 3 fichiers disponibles sur [REF2](https://github.com/AghaMS/Multidimensional_Knapsack_Problem_Modelling), ou chaque fichier est l’équivalent d’un problème d’optimisation a 10 sacs à dos et 10 contraintes, mais les solution ne sont pas encore calculés.

Il est bien clair qu’il ne sera pas possible d'entraîner et tester le modèle sur uniquement 3 instances (fichiers) donc nous allons décomposer chaque problème en plusieurs sous-problèmes au but d’augmenter la taille de nos données (le nombre d'instances).

Pour mieux expliquer, le troisième fichier par exemple est un problème à 500 objets, nous allons le décomposer en 100 **(500 / 5)** sous problemes de 5 objets en divisant les capacités des sac a dos aussi par 100, car si on gardait les même capacités, la solutions serait toujours de retenir tous les objets et ca ne devient plus un problème d’optimisation combinatoire.

Ensuite il faut calculer la solution relative à chaque instance, et pour cela nous allons utiliser l’outil Cplex d’IBM.

Cette idée a été tirée de [REF3](https://towardsdatascience.com/the-binary-multidimensional-knapsack-problem-mkp-2559745f5fde).

Une fois la décomposition de chaque fichier et le calcul des solutions sont achevés, il ne reste qu'à transformer les instances en matrices de la forme suivante:



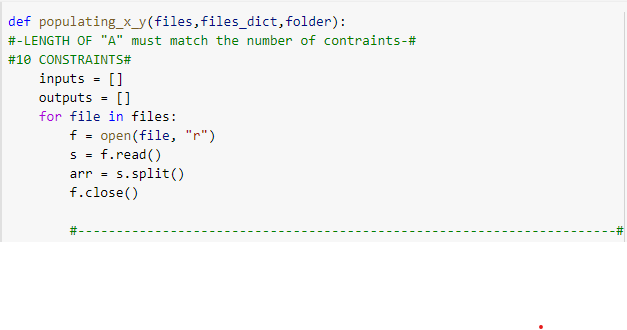
Où:

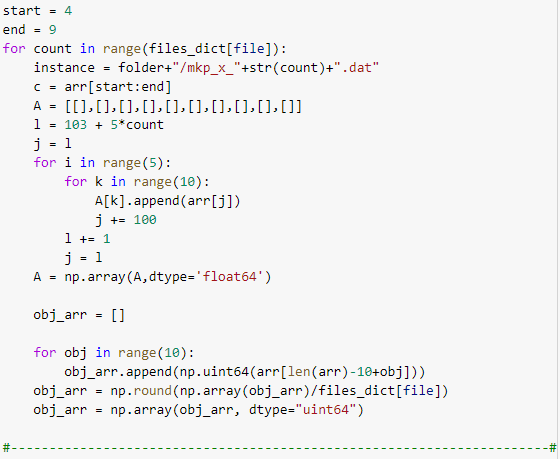
* **P1**: Profit du premier objet, **P2**: Profit du deuxième objet,
* **C1,1**: valeur de la première contrainte du premier objet,
* **C1,10**: valeur de la dixième contrainte du premier objet,
* **C2,1**: valeur de la première contrainte du deuxième objet,
* **C2,10**: valeur de la dixième contrainte du deuxième objet,
* **Cs1**: capacité du premier sac à dos,
* **Cs10**: capacité du dixième sac à dos.

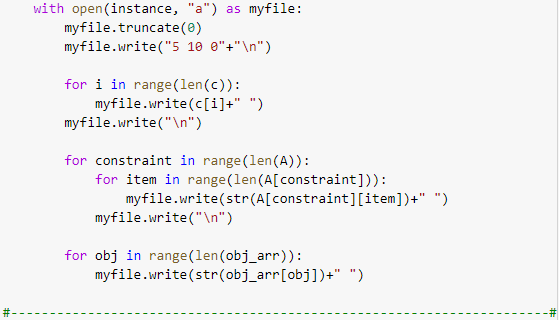
La solution relative à cette instance doit avoir la forme suivante:

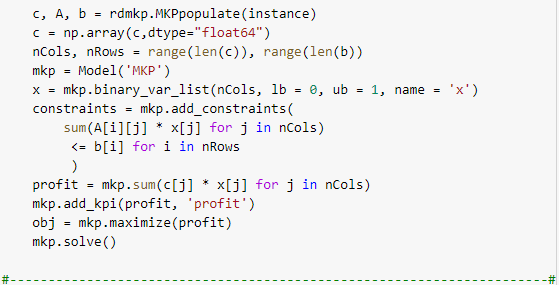
Où **V1, V2 … V5** sont successivement les variables binaires de décision de rejet ou de rétention de chaque objet.

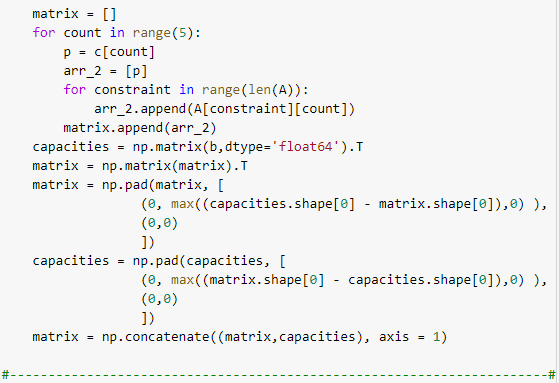
Les captures ci-dessous représentent le code de la fonction permettant d’achever ce résultat:

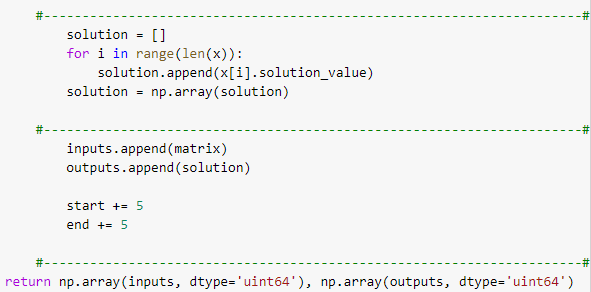




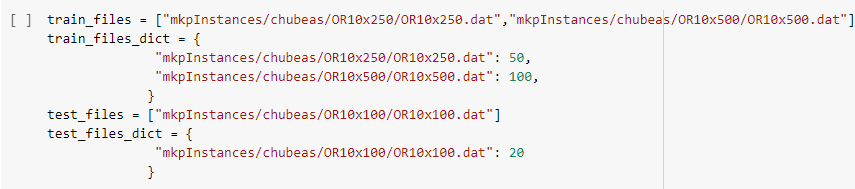


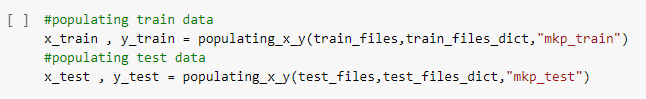






La fonction **Populate\_x\_y** nécessité d'être appelée dans une boucle pour appliquer le traitement sur les 3 fichiers comme suit:



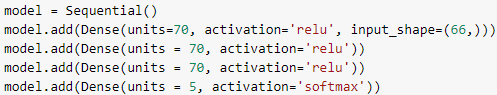


On obtient ainsi 150 instances d'entraînement et 20 instances de test.

Nous avons maintenant tous les atouts pour passer à la partie modélisation.

# 3.2 Modélisation

Comme déjà mentionnée dans la partie **(2.3),** le but de ce projet n’est pas d’élaborer un modèle de l’état de l’art, nous allons utiliser la même architecture que celle mentionnée dans **(2.3)** nous allons seulement changer la forme d’entrée du modèle et le nombre d’unités (neurones) de chaque couche pour que nos données y soient compatibles en exécutant le code suivant:



En exécutant la fonction model.fit, notre modèle s'entraîne sur les données générées dans la partie précédente pour achever une précision de ~58%.



# 4. Conclusion

Pour clôturer, l’idée d’utiliser l’**intelligence artificielle (IA)** pour résoudre les problèmes d'**optimisation combinatoire (OC)** au lieu des algorithmes déterministes parvient du fait que ces derniers deviennent très lents et complexes du point de vu temps d'exécution quand le nombre d’itérations nécessaire est très important, le coût augmente exponentiellement en fonction du nombre de combinaisons possibles.

Le lien entre l’**IA** et l’**OC** est un sujet d’actualité et fait l’objet de plusieurs recherches, mais il y a très peu d'implémentations de ceci surtout quand il s’agit du multi-objectifs.

Ce travail est donc une initiative pour concrétiser la théorie derrière ce lien et qui reste un sujet à critiquer et optimiser.

Il est possible d'améliorer la démarche, la qualité et la taille des données et le paramétrage du modèle prédictif.