Nettoyage des données pour la classification à l’aide des réseaux de neurones

I - Introduction

Nous sommes au 21e siècle la révolution pétrolière et électronique est passé. Comme certains experts le disent nous sommes à l'ère du Big Data. Les données sont pour beaucoup l'or des 21 e siècle. Il est donc nécessaire pour tout scientifique de comprendre ceux-ci pour obtenir des résultats satisfaisants. Contrairement à ce que nous pourrions penser, les données sont omniprésentes et sous diverses formes. Ceci dit, trouver dès la première fois des données entièrement exploitables n'est qu'un mythe. En tant que scientifiques, nous devions donc veiller à l'intégrité et à la propriété des données avant de les utiliser.Avec l'explosion du Machine learning et notamment du Big Data, cela devient encore plus véridique. Ceci dit les scientifiques ont pris pour défis la mise en place de techniques visant à rendre les données propres et exploitables. Toutefois bien que le nettoyage de données est une étape cruciale avant la mise en place d'un algorithme de Deep Learning, **Ne serait-il pas possible d'effectuer ce nettoyage avec du deep learning ?**

II - Concept important

1. Cycle de vie d’un projet de Data Science

Pour bien comprendre le processus de nettoyage des données, il faut comprendre tout d’abord comment se déroule un projet de **Data Science.** Un projet de Data Science se déroule notamment en 4 grandes étapes que sont:

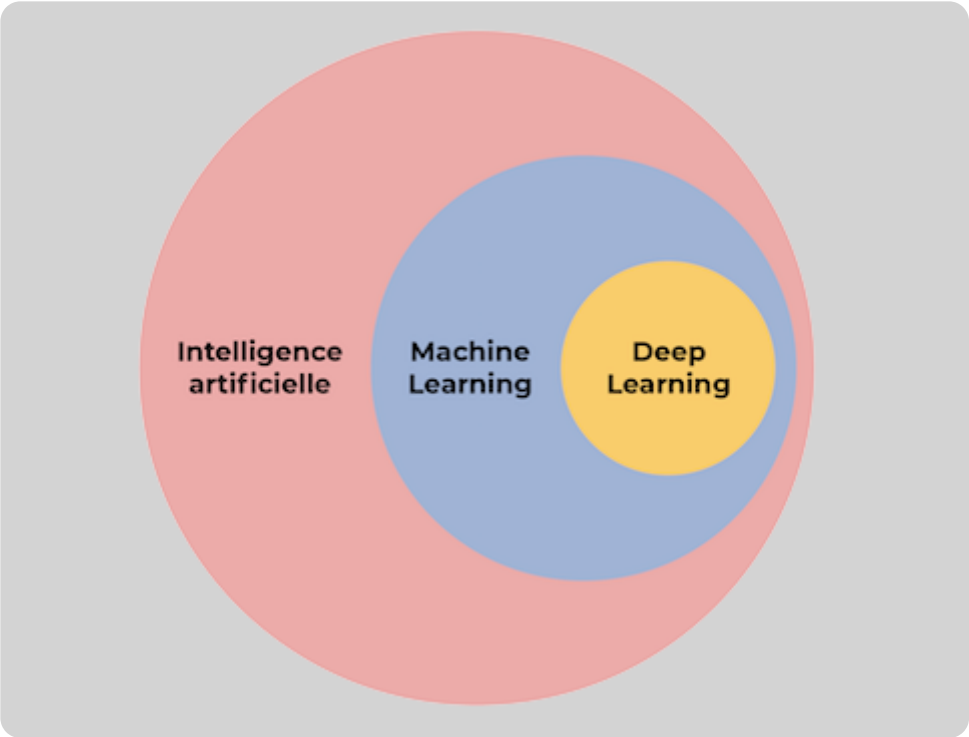
- La collection des données
- La préparation des données
- L'exploration et la visualisation des données
- L’expérimentation et la prédiction des donnés

Le nettoyage des données intervient à la phase **de préparations des données**.

2. Quesque le machine Learning ?

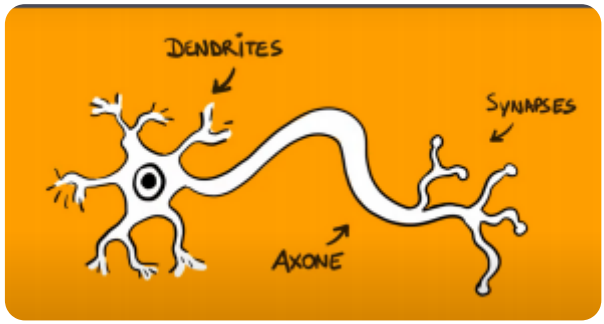
Unanimement, le ML est un sous-domaine de l'intelligence artificielle que l'on peut lui-même définir comme une imitation de l'intelligence humaine par une machine. Pour certains, le machine learning peut être défini comme un moyen d'adapter l'ordinateur aux situations pour lesquelles il n'a pas été explicitement programmé.

3. Le deep Learning



Quesque le Deep learning ?

Le **Deeplearning** est l'une des techniques de ML qui a révolutionné le 21e siècle. Son concept est fondé sur la reproduction de la structure cérébrale humaine. Ce dernier comprend des milliers de neurones connectés par des synapses. Nous allons donc faire quelques neurosciences pour comprendre comment ils fonctionnent et de la même manière comment Deeplearning fonctionne.



Structure d’un Neurone Humain

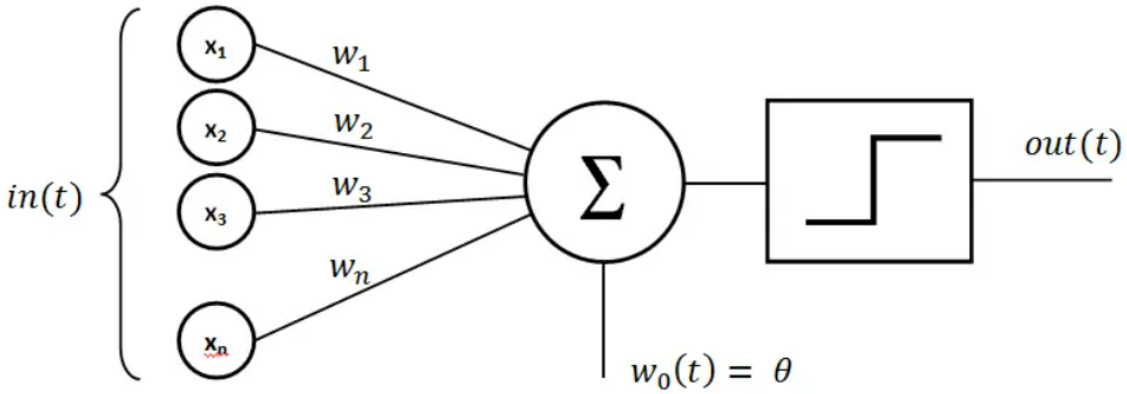
Comment fonctionne un neurone Biologique ?

Il faut tout d'abord remarquer que le neurone biologique est constitué de trois grandes parties que sont: les `dendrites`, `l'axone` et enfin `la synapse`. Le neurone reçoit un signal d'activation par sa **dentrites.** Ensuite ce signal passe par **l'axone** du neurone puis active d'autres neurone par le biais de sa **synapse**. Ceci est souvent connu sous le nom de la **transmission synaptique**. Alors comment ce mode de communication a influencé le développement du Deep Learning?

Fonctionnement d'un réseau de Neurone Artificiel

La technique à la base du succès du Deep learning est le `Réseau de Neurones Artificielle` . Celle-ci repose sur la structure des réseaux neuronaux, d'où leur nom. Il faut toutefois savoir que cette technique n'est pas liée au Deep Learning, la spécifiée du Deeplearning vient du fait qu'il peut avoir plusieurs couches de neurones intermédiaires entre l'entrée et la sortie d'où l'appellation de cette technique.

La structure fondamentale d'un réseau neuronal est un neurone (`Naturally` ☑). À la différence d'un neurone biologique, il aura les éléments suivants:



La formule du Perceptron

- **Des poids:** Un réseau de neurones est généralement représenté comme un **graphe pondéré** où les noeuds sont les neurones. Pour chaque donnée entrant dans le neurone, une **branche(poids)** d'une valeur donnée est associée à celle-ci. Le réseau de neurones consistera à trouver les meilleures pondérations pour chaque liaison afin de minimiser les différences entre les prédictions du réseau et les résultats concrets.
- **Un biais:**En plus de la pondération, chaque neurone se voit attribuer un **biais** qui cette fois est indépendant des intrants (Les données). Le réseau neuronne va également chercher à trouver les meilleurs poids pour chacun neuronne.
- **Une fonction d'aggrégation:** Cette fonction prend toutes les entrées x_i d'un neurone et les multiplie par le poids w_i correspondant à ces entrées, puis ajoute le biais neuronal.

$$f = \sum(\pm 1) w_i * x_i + b$$

- **Une fonction d’activation:** Comme son nom l'indique, la fonction d'activation est une formule mathématique activée sous certaines conditions. Lorsque les neurones **agrègent** les valeurs, elles sont transmises à la fonction d'activation, qui vérifie si la valeur calculée est supérieure au seuil requis. Il existe plusieurs fonctionnalités d'activation, parmi les plus populaires sont:
  - **Sigmoïde**: produit une courbe en forme de S. Bien qu'elle ne soit pas linéaire, elle ne tient pas compte de légères variations dans les intrants, ce qui donne des résultats semblables.

$$a(z) = \frac{1}{1 + exp(-z)}$$

  - **Fonctions de tangente hyperbolique (tanh):** Il s’agit d’une fonction supérieure comparée à la fonction Sigmoid. Toutefois, elle tient moins compte des relations et sa convergence est plus lente.

$$a(z) = \frac{exp(z) - exp(-z)}{exp(z) + exp(-z)}$$

  - **Unité linéaire rectifiée (ReLu):** Cette fonction converge plus rapidement, optimise et produit la valeur souhaitée plus rapidement. C’est de loin la fonction d’activation la plus populaire utilisée dans les couches cachées. D’une manière simple elle renvoie 0 pour les nombres négatifs et la valeur du nombre pour les nombres positifs

$$a(z) = max(x, 0)$$

  - **Softmax:** utilisé dans la couche de sortie car il réduit les dimensions et peut représenter une distribution catégorique. Cette fonction reçoit un vecteur $z = (z_1, ..., z_K)$ et pour chaque composant calcule sa valeur comme suit:

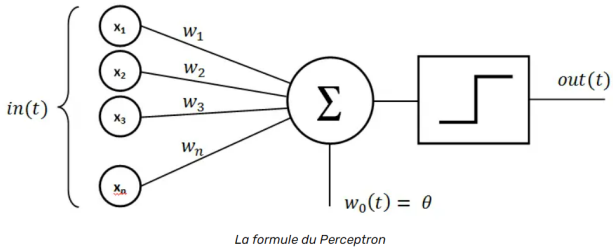
$$a(z)_j = \frac{exp(z)_j}{\sum_{k=1}^{K} exp(z)_k}$$

Type de réseaux de Neurone

En fonction du type d’usage, il existe une multitude réseaux de Neurones. Toutefois nous allons nous attarder sur quatre grands types de réseaux de Neurones que sont:

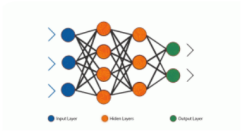
- **Perceptron Multicouche (MLP)**
  Un perceptron est le réseau neuronal le plus simple à construire. C'est un neurone unique qui agit comme une régression logistique. En effet pour une entrée donnée il fournit une sortie binaire. Un perceptron Multicouche sera comme son nom l’indique formé de multiples perceptrons sur plusieurs

couches. Ils peuvent effectuer bon nombre de tâche comme classifier des images, des données tabulaires ou même textuelles. Toutefois leur manque de spécialisation est un inconvénient ce qui nous amène aux réseaux de Neurones suivants.



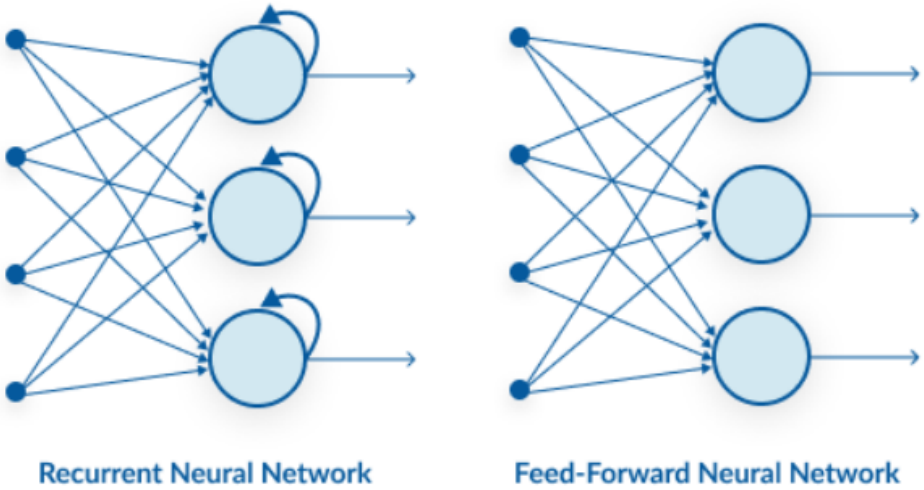
La formule du Perceptron

Structure d'un Neurone Humain



Perceptron Multicouche

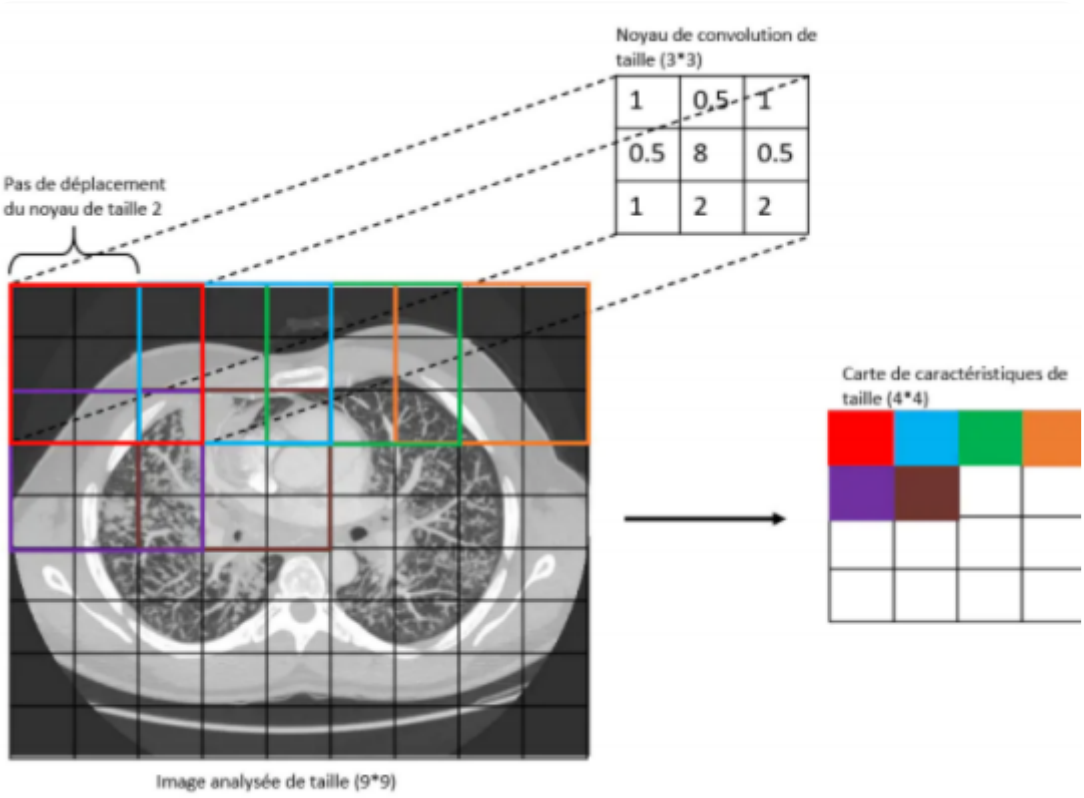
- **Réseaux de Neurone Récurrent (RNN)**
  Dans un perceptron multicouche le résultat d'une couche de neurone est transmis à la couche suivante car chaque couche possède des paramètre différent. Toutefois dans un Réseaux de Neuronne récurrent, Un neurone peut conserver en mémoire un résultat passé pour en produire un autre résultat (d'ou la boucle sur le shéma). On peut utiliser ce type de réseaux pour des séries temporelle, des données textuelle ou même des données audio.

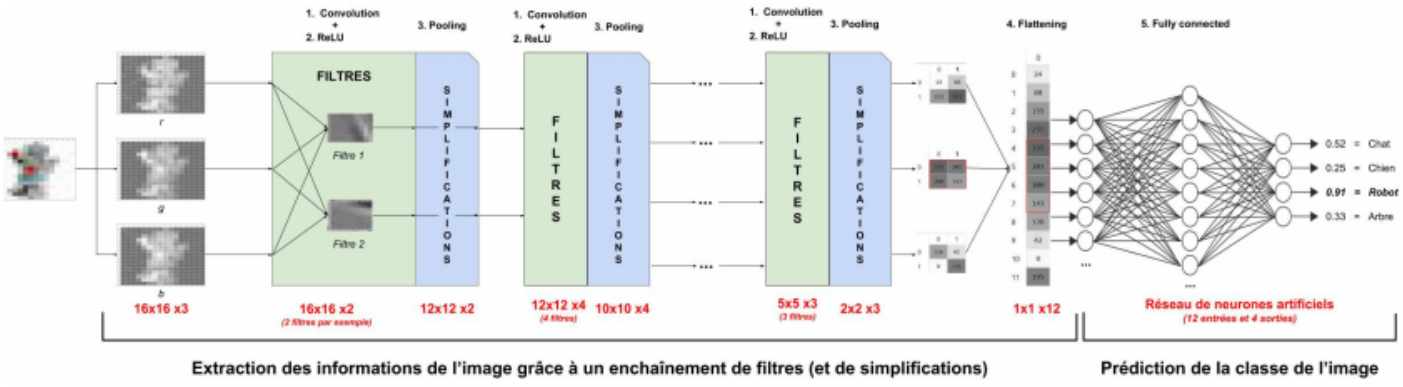


Différence ANN et RNN

- **Réseaux de Neurone Convolutionnel (CNN)** Lorsqu'on veut entrainer un réseau de Neurones traditionnel à base d'image, il existe une multitude d'opération de pré-processing à effectuer avant de pouvoir entrainer les réseaux. L'avènement des CNN a contribué drastiquement à la réduction de ces opérations. En effet ce type de réseaux possède des couches spécialisées pour effectuer les opérations de pré-processing. Sans entrer dans les détails, voici les couches principales :
- **couches de convolution (CONV)**

- **couches de correction (ReLU)**
- **couches de pooling (POOL)**





Algorithme de descente du gradient avec Momentum

Lorsqu'un réseau de données est entrainé sur un dataset **X** pour l'évaluer on peut lui définir une fonction qui va représenter l'écart entre les prédictions du réseau et les valeurs réelles du dataset. L'apprentissage du neurone va consister à minimiser cette fonction coût. Pour ce faire on utilise généralement la méthode de l'algorithme du gradient. Toutefois celle ci peut tomber sur un minimum local de la fonctions ce qui auras pour effet d'avoir un modèle sous entrainé. Pour pallier à ce problème les scientifiques on développé un algorithme de gradient evolué avec Momentum. Un **momentum** peut être décrit en physique comme un vecteur quantité de mouvement $\vec{p} = m * \vec{v}$ ou m est la masse du solide et $\vec{v}$ son vecteur vitesse. Le principe seras d'utiliser pour l'itération i les informations utilisé à l'itération $i-1$. En effet dans un algorithme de gradient classique, à chaque itération on a:

$$x_{k+1} = x_k - \alpha * \nabla f(x_k)$$

Dans le gradient avec **momentum** on aura:

Soit C_k tel que $C_{k+1} = \alpha * \nabla f(x_k) + \beta * C_k$ avec α le **learning rate** et β le **coefficient momentum** tel que $0 \leq \beta \leq 1$.

$$x_{k+1} = x_k - C_{k+1}$$

Ainsi on remarque que à chaque itérations i la méthode conserve les informations de changement C_k des itérations précédentes.

4. Apprentissage par renforcement

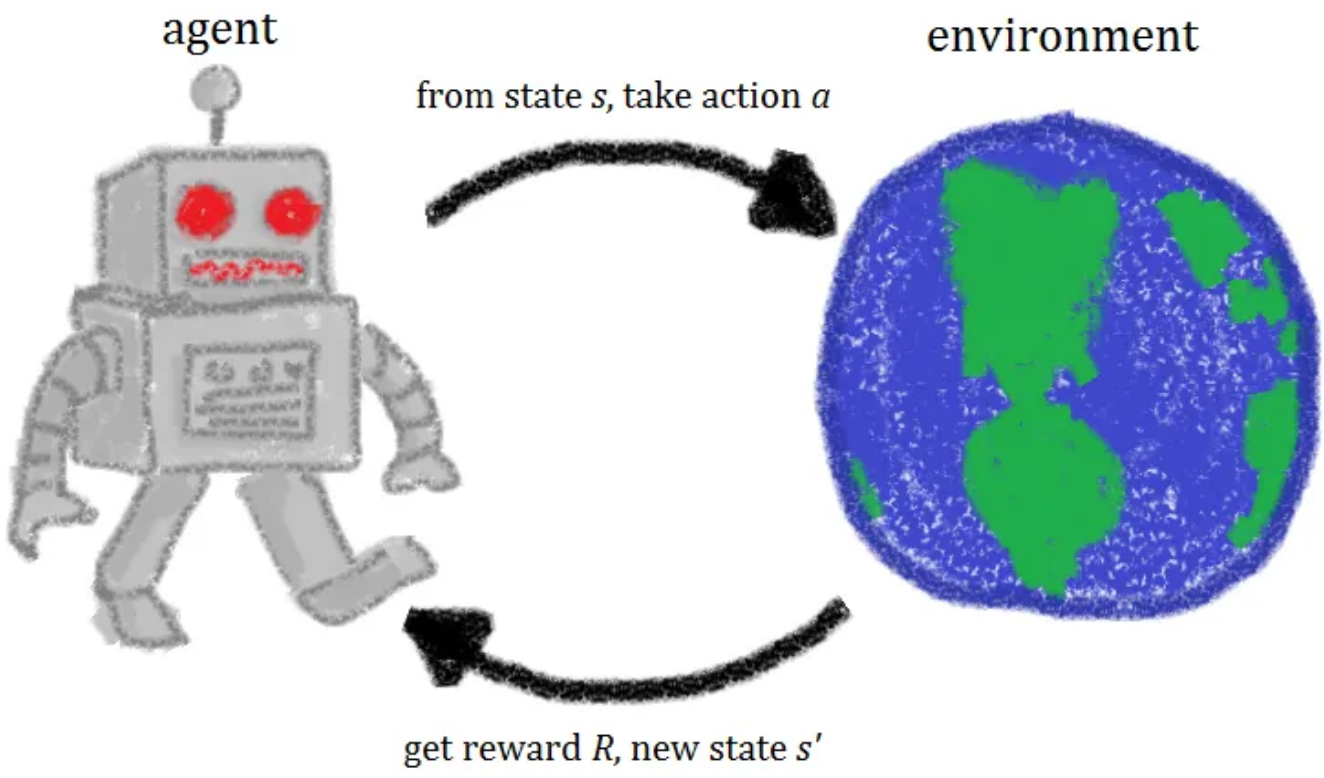
L'une des techniques de transformation des données par deep learning que nous verrons plus tard est effectuée par un algorithme de renforcement. Nous allons essayer de comprendre comment fonctionne ce genre d'algorithmes.

Quesque l'apprentissage par renforcement ?

Contrairement au **algorithmes supervisé ou non supervisé** un algorithme par renforcement est un algorithme interactif dans lequels un algorithmes éssaie plusiseurs tentatives et est recompensé lorqu'il effectue une tentative. Le but pour cet algorithme seras donc de maximiser sa récompenses. Ce type d'algorithme est à la base du célèbre programme **AlphaGo** qui as battu le champion du monde en Go.

Comment fonctionne t'il?

Dans ce type d'algorithme on distingue deux termes principaux: **L'agent** et **l'environnement. L'argent** est l'algorithme qui vas se charger de l'apprentissage tandis que **l'environnement** est le monde dans lequel vas vivre et interagir l'agent. L'agent peut interagir avec l'environnement par le biais de certaines action toutefois il ne peut pas le modifier. Dans une analogie avec le monde réele un **agent** peut être l'homme tandis que **l'environnement** la nature. Le dévéloppeur programme cet environnement de telle sorte à ce que l'agent soit recompensé pour une bonne action et pénalisé dans le cas contraire. Dans un environnement un agent à un ensemble d'action permise qu'on nomme **l'action space (Espace d'action).** Celui ci peut être continu ou discret **.** Lorsqu'une action est effectué par l'agent, un nouvel état de l'environnement lui est retourné **(state)**



III - Etude des problème liée à la proprieté des données

Avant de voir quelque techniques de nettoyage de données à base de deep learning, nous allons éssayer de voir les problème les plus fréquent auquelle on peut faire face en phase de préprocessing des données.

A - Type de Données manquantes

Il existe suivant la répartition des valeurs manquantes plusieurs types de données manquantes. On distingue entre autre

- **MCAR** → **M**issing **C**ompletly **A**t **R**andom: Ces types de données signifie que Les valeurs manquantes sont indépendantes des observations. Par exemple si on recueilles des données chez diverses étudiants et que certaines données sont perdu on peut les considérer comme complètement aléatoire car cela n'a rien à avoir avec l'étudiant en soit. En terme probabliste soit M_i ***l'évènement '**Données manquante pour l'entrée i**' et** D** les données de l'entrée i

$$P(M_i/D_i) = P(M_i)$$

Toutefois il est difficiles de former ce genre d'affirmation car des données totalement aléatoire sont rare en réalité

- **MAR** → **M**issing **A**t **R**andom: Contrairement au noms, une donnés qui est **MAR** n'est pas manquante du au hasard. En effet cette abscence de données peut être expliqué par d'autre variable observé mais pas par la valeur spécifique de la variable manquante. Pour notre exemple du sondage sur les étudiants on remarque que les étudiants de classes prépa on plus tendances à ne pas répondres à une question données d'ou la valeur manquantes de celle ci. Ainsi notre valeur manquante peut être expliqué par la fillière de l'étudiant.
- **NMAR** → **N**ot **M**issing **A**t **R**andom: Pour finir, ce type de données est relatif à la valeur de la variable manquante elle-même. Par exemple pour une interview sur le revenu de certains étudiants, certains n'ont pas voulu répondre car ils ont un très faible revenu (valeur logique de la variable)

B - Contrainte sur les données

1. Contrainte de types de données

Généralement lorsqu'on travaille avec les données il est préférable que toute les données d'une même colonne soit du même type. Par exemple une colonne qui contient des entiers ne doit pas contenir de chaine de caractère au risque d'impacter les résultats de calcul. De plus Il est préférable de travailler avec des données catégorielle aux maximum. En effet cela améliore grandement les performances car chaque valeur de la catégorie est connu à l'avance

Solution:

- Convertir les données dans un type commun
- Pour les données textuelle on peut essayer de les regrouper en un type catégorielle

2. Contraintes d'intervalle

Pour les données numériques, il est nécessaire de vérifier qu'elles sont dans un bon intervalle. Par exemple, une température de 100 degrés Celsius indique un résultat anormal. De même, pour les valeurs catégorielles, il faut s'assurer que toutes les valeurs se trouvent dans l'ensemble catégoriel. Par exemple un groupe sanguin **Z+** n'a aucun sens.

Solution:

- Supprimer les lignes contenant les valeurs abberantes
- Remplacer ces lignes par une valeurs données (Imputation)

3. Contraintes d'unicité

Dans nos données on doit S'assurer de l'unicité d'une partie des attributs. Par exemple si on collecte des informations sur des étudiants d'une classe, on ne doit pas avoir un étudiant avec le même numéro d'étudiant, le même nom et le même prénom.

Solution:

- Conserver une seule données
- Fusionner les données

C - Problèmes des données textuelles et catégorique

Comme mentionnée plus haut, Il est préférable d'utiliser des données catégorielles à la place des données textuelle ou tout autre type de données si possibles. Toutefois lors de cette conversion on peut rencontrer certains problèmes.

1. Consistence des valeurs

Lorsque les données sont sous forme textuelle, on peut avoir pour une même valeur plusieurs forme, ce qui peut conduire à une inconsistance de données. Par exemple dans un dataset les mots `**Lundi**` et

`lundi` seront considéré comme différent ou encore `Bonjour` vas être différent de `Bonjour `

2. Similarité de chaine de caractère

De même que pour la consistance des chaines de caractère, deux données bien que différent au premier abord peuvent en réalité réprésenter le même élément. Par exemple on peut avoir pour l'équipe du maroc les deux enrégistrement suivant: `Equipe Marocaine` , `Equipe de football Marocaine` on l'air assez différents mais on peut remarquer qu'il réprésente le même mot. Ainsi les considérer comme deux catégories différentes peut être préjudiciable.

Solution

La résolution de ce problème passe par la comparaison de similarité entre chaine de caractère. Pour ce faire on peut utiliser plusieurs métrique comme:

- La distance de Damerau-Levenshtein
- La distance de Levenhstein
- La distance de Hamming
- La distance de Jaro Distance

Il faut noter que tous ces problèmes de données textuelles et catégorique sont également connu sous le nom d'**Entity Matching** .

D - Complétude (Valeur Manquante)

L'un des plus grand problème de nettoyage des données est la **complétude.** Celui ci intervient lorqu'on as des données manquantes dans notre dataset. Cela peut être de diverse origine: capteur défaillant, valeur exhorbitante, ou encorre erreur humaine. Toutefois la pluspart des algorithme de machine learning et notamment de deep learning ne travaille pas avec les valeurs manquantes. Il devient donc nécessaire de les remplacer par une valeur adéquate. Pour ce faire on aura plusieurs approche

Approche de Solution

- **Suppression des enrégistrements avec des valeurs manquantes:** La solution la plus simple mais également la moins recommandé car lorque le nombre d'enrégistrement ayant des valeurs manquantes augmente cela peut entrainer d'énorme perte de données et donc de précision dans notre modèle de machine learning
- **Imputation:** Cette méthode nécéssite un connaissance du domaine pour choisir le meilleur estimateur pour une valeur données. Bien que plus compliqué elle est recommandé car elle permet de conserver nos données. On distingue plusieurs méthode d'imputations. On peut citer entre autre:
  - **Imputation univarié:** Durant cette méthode on vas utiliser uniquement la variable mère pour prédire les entrées manquantes. On peut par exemple prendre pour les valeurs manquantes la moyenne de la variable mère(Variable numérique) ou le mode (variable catégorique)

$$x = \overline{x}$$

- **Imputation multivarié:** Dans ces méthode on vas utiliser d'autre variable pour prédire les variable manquante. On peut par exemple pour une variable y utiliser la regression linéaire avec une ou plusieur autre variable puis ainsi estimer la valeur manquante

$$y = aX + b$$

- **Hot - Deck:** C'est un ensemble de méthode d'imputations consistant à remplacer les valeurs manquantes d'une observation par ceux de populations similaire observé. Pour ce genre de méthode on peut utiliser un Algorithme de KNN-Neighbor

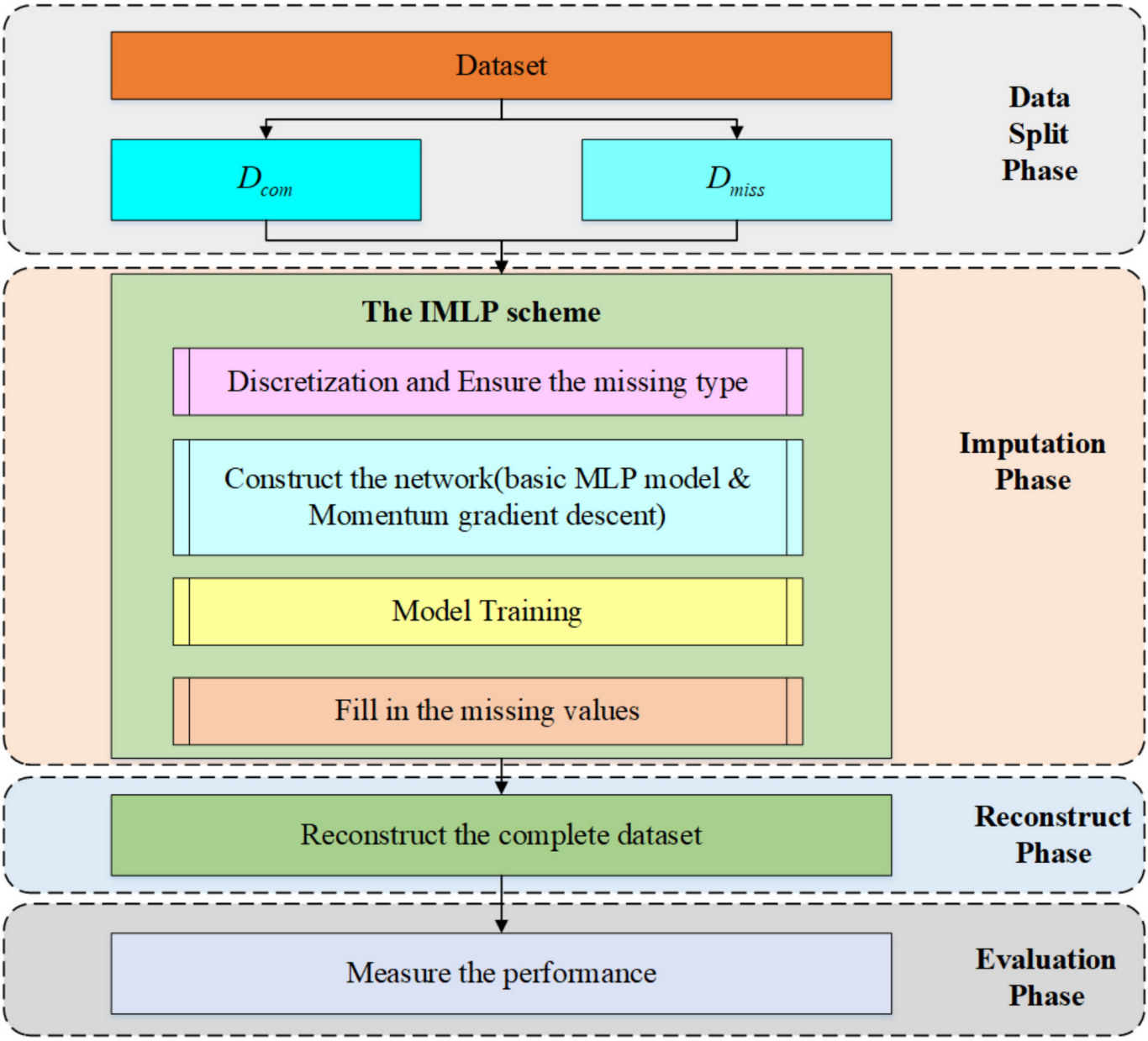
IV - Technique de nettoyage avec le deep learning

A. Imputation de valeur discrete à base de perceptron Multicouche Amélioré

Pour effectuer l'imputation bon nombre de méthodes existe comme nous avions pu le voir On peut avoir entre autre l'imputation par mode, KNN, autoencoder et autre.Toutefois ces méthodes offre généralement de très bon résultats lorsque les valeurs manquantes sont des valeurs continue. Hors il est tout autant important de travailler avec les valeurs manquantes d'une distribution discrète.

Apercu de la méthode

Pour imputer les valeurs manquantes on vas utiliser une technique à base de perceptron Multi Couche comme décrite sur l'image suivante.


Dataset
D_{com}
D_{miss}
Data Split Phase
The IMLP scheme
Discretization and Ensure the missing type
Construct the network(basic MLP model & Momentum gradient descent)
Model Training
Fill in the missing values
Imputation Phase
Reconstruct the complete dataset
Reconstruct Phase
Measure the performance
Evaluation Phase

Etape 1: Diviser le dataset en deux parties:

- D_{com}: Le Dataset dont les valeurs ne sont pas nulles
- D_{miss}: Le Dataset avec des valeurs manquantes

Etape 2: Discretiser les valeurs en utilisant un algorithme de discrétisation (comme le one-hot-encoding) puis les faires entrer dans le `IMLP` (MLP amélioré avec algorithm de descente de gradient avec momentum) qui vas remplir les valeurs manquantes

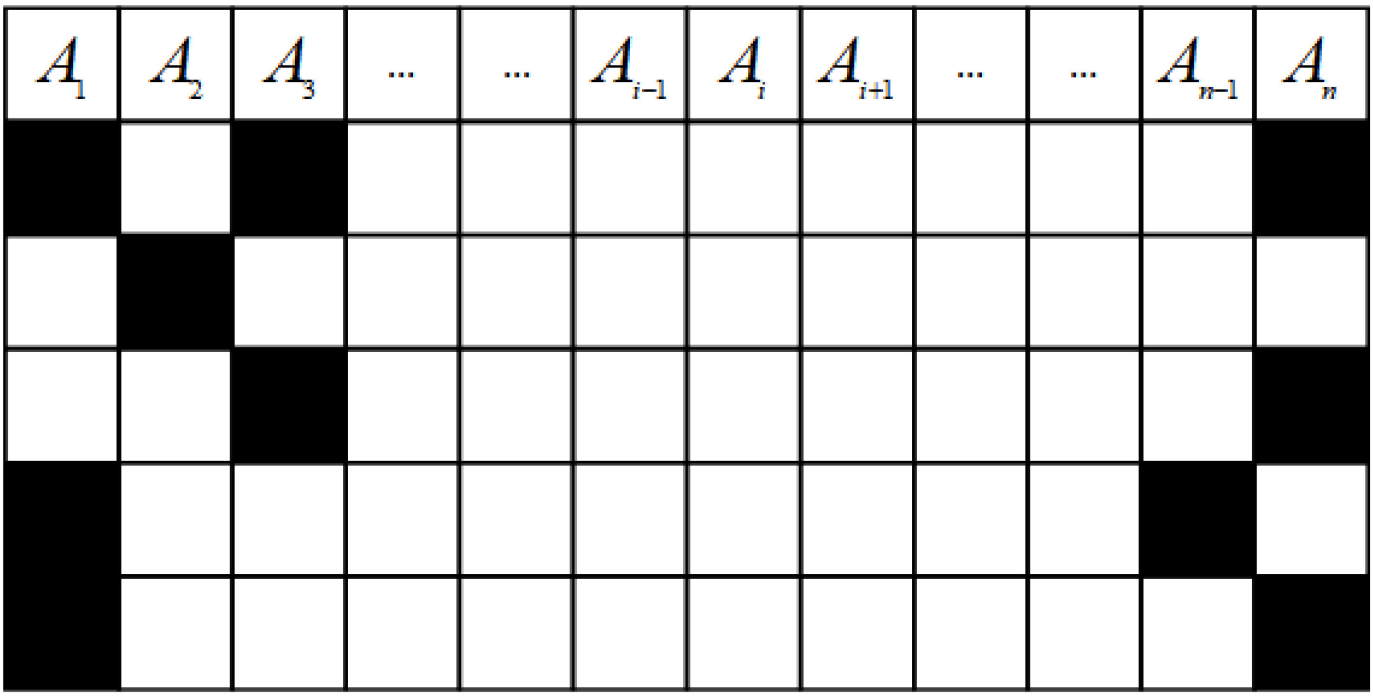
Etape 3: Recombiner D_{com} et D_{miss} pour retrouver le dataset original. On note que D_{miss} seras à cette étape remplis avec des valeur estimé par l'algorithme

Etape 4: Evaluer les performances du modèle

Explication détaillé de la méthode

1. Détermination des types avec valeur manquantes

Soit D notre dataset. On a D_{miss} qui correspond au datasets avec les entrées manquantes et D_{com} le datasets avec les données complètes. Dans le datasets manquants, on aura $A_i (i = 1..n)$ ou n est le nombre de feature du dataset. Dans la grille suivante on peut voir une modélisation du Dataset ou les cases noires correspondent au données manquantes.



Pour chaque ligne i on vas noter les types manquants mt_i. Par exemple suivant notre shémas, $mt_1 = 1, 2, n$. On obtient ainsi l'ensemble des types manquants $MT = mt_i \ i \in (1..m)$ ou m réprésente le nombre d'enrégistrement de D_{miss}

2. Constructions du IMLP

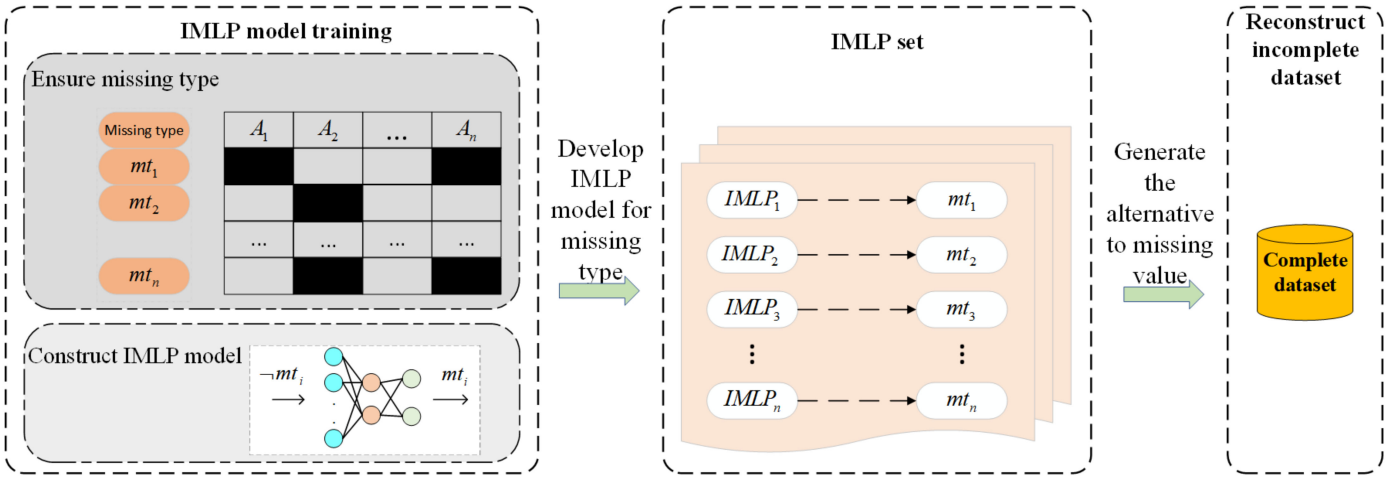
Pour notre réseaux de neurone la couche d'entrée vas prendre l'espace des enrégistrement $X = x_1, x_2, ..., x_m$ ou x_i correspond à un enrégistrement du Dataset. De même la couche de sorties vas contenir n neurone qui vont chacun correspondre à une **feature** des données. Les couches caché vont utiliser une fonction d'activation $relu(x)$ tandis que la couche de sortie vas utilser une foncton $sigmoid(x)$. Le IMLP ne pouvant pas traiter directement avec les valeurs manquantes on vas les pré-remplir avec une valeur données.

3. Entrainement du modèle

Avant l'entrainement du modèle on doit s'assurer que les valeurs discrètes du modèles (Valeurs textuelle, Entier) soit discrétiser. On peut par exemple utiliser la technique du **one-hot**. Cela vas permettre d'obtenir des valeurs discrètes pour la prédictions. Ensuite on vas se servir de D_{com}. De plus on vas diviser D_{com} en train-set et test-set pour effectuer l'entrainement. Pour D_{miss} on vas faire entrer dans le modèle les valeurs présente plus les valeurs absentes seront déduites

4. Reconstruction du datasets incomplet

Durant cette phase le but seras d'utiliser le IMLP entrainé pour prédire les valeurs manquantes de D_{miss} .La première étapes seras de déterminer les types manquants du datasets (mt_i). Les données manquantes seront remplis une par une pour enfin générer un dataset complet.



B. Transformation automatique des données à bases de Deep Reinforcement Learning

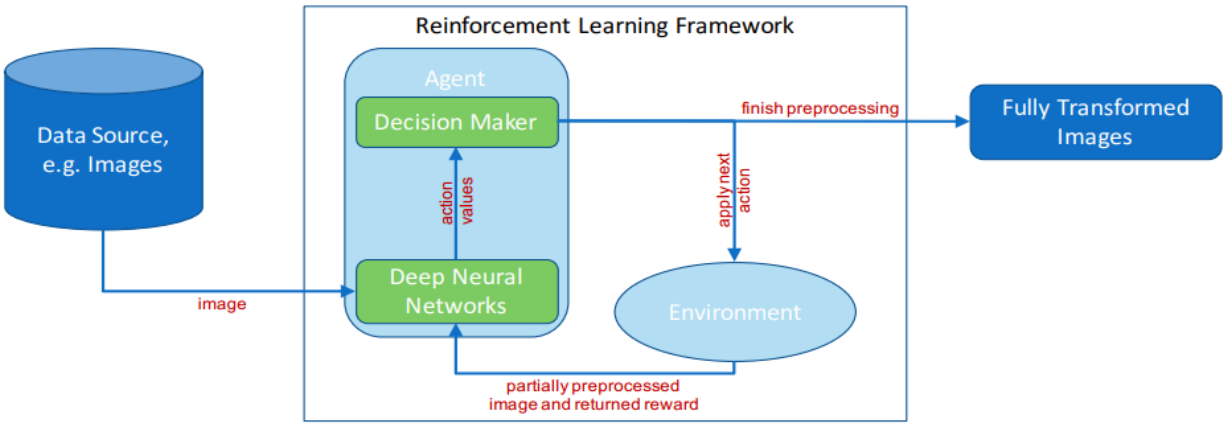
Avant d'entraîner un modèle il est souvent nécessaire d'effectuer un certain nombre de transformations. Par exemple pour une image, il peut être nécessaire de filtrer, renverser ou couper ces images avant de pouvoir les faires passer aux réseaux de de Neurone. Toutefois ces transformations bien que répétitives sont toujours faites par intervention. Dans la perspective de notre étude nous allons voir une technique à base de Reinforcement learning à base de Deep Learning qui va permettre d'appliquer directement des transformations données à un ensemble de données.

Motivation

Bon nombre de scientists se sont penché sur l'automatisation de la transformation des données.Toutefois la plupart de ces études sont méné sur un nombre relativement limité de transformation comme la **normalisation**, **standarisation** ou **dicretisation.**. De plus ces études applique les transformations sur tout l'ensemble de données plutot que d'appliquer une transformation spécifique. En effet la transformation des données réprésente bon nombre de challenge. Pour une donnée D et deux transformation T et P , il est important de remarquer que $T(D) + P(D) \neq P(D) + T(D)$ car l'ordre des transformation est important. De plus avec un nombre important de transformation le temps d'apprentisage seras énormes. De plus comme vu précédemment appliquer le même ensemble de transformation à tous un Datasets peut être innéficaces car chaque données à sa spécifité. Par exemple deux images ayant été pris d'un différents angle et qui recoivent une rotation de 90 dégré n'auront aucun sens. Enfin une mauvaise transformation peut totalement affecter le sens de la donnée. Par exemple une rotation de 180 dégré sur le chiffre **9** vas donner le chiffre **6**.

Apercu du framework

Comme tout algorithme de Reinforcement learning on aura un **agent** et un **environnement**. Dans notre cas l'agent seras un réseaux de neuronne dont la sorties seras un ensemble de transformation qui seront utilisé par un chef d'orchestre qui vas décider si l'image nécessite plus de transformation ou peut être considéré comme correctement transformé. Lorsque le chef d'orchestre décide que l'image nécessite d'être transformé, l'environnement vas se charger de cette transformations puis vas renvoyer l'image dans le réseaux de neuronne pour une nouvelle évaluation. Ce processus seras répété jusqu'a ce que le chef d'orchestre juge que l'image est correctement transformé. De plus pour que l'agent apprenne , l'environnement vas évaluer l'impact des transformations effectué puis envoyer une récompenses basés sur celle ci. Ce processus est resumé dans l'image suivante.



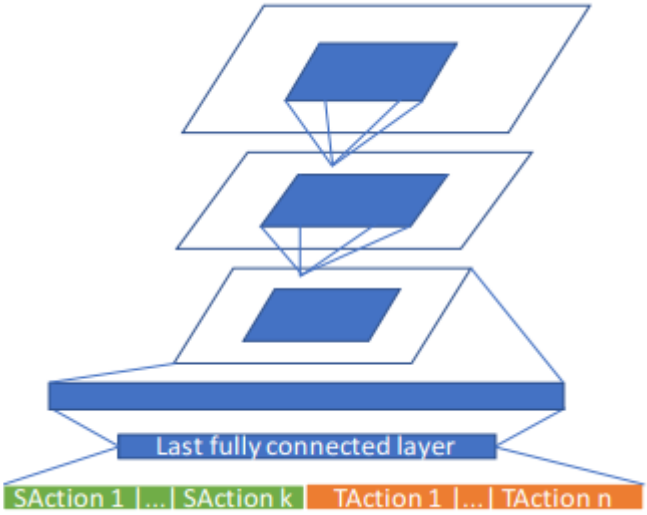
1. Espaces d'action et Etats

1.1 Espaces des Etats

Les états dans notre frameworks seront les images. L'état original vas correspondre à l'image d'origine. Après chaque transformations un nouvel états seras retourné par l'environnement

1.2 Espaces des actions

Pour notre étude un **Deep Q-Network (DQN)** seras utilisé. La couche de sorties vas être l'espaces des actions et seras constitué de deux types. La première parties est un vecteur logique qui vas contenir la probablité d'appartenance à la classe k, elle est nôté $S_{actions} = (S_{actions_1}, S_{actions_2}, ...S_{actions_k})$. Ce type d'action vont être des actions d'arrêts en effet si le chef d'orchestre choisi une action $S_{actions_i}$ alors le processus vas s'arrêter et vas prédire la classes i pour l'image. La seconde actions quand à elle, vas contenir une ensemble de n transformations $T_{actions} = (T_{actions_1}, T_{actions_2}, ...T_{actions_n})$. Si une transformation i doit être effectué le chef d'orchestre vas choisir une action $T_{actions_i}$. Ces deux ensembles d'actions forme ainsi un Espaces d'actions de tailles $k + n$. On remarque que la première partie de cet ensemble est adapté à la tache du réseau (dans notre cas une classification).



1.3 Restaurations des transformations

Pour chaque transformations A appliqué il est important d'avoir une transformation $-A$ qui peut lui être appliqué pour annuler l'effet de la transformation A. En effet si cette tranformations conduit à un résultat sous performant il peut être interressant de l'annuler. Pour des transformations complexe cela peut conduire à une grande consommation de mémoire. Toutefois on vas utiliser un méchanisme pour controller le nombre maximum de transformations pouvant être appliqué.

2. Chef d'orchestre

C'est l'une des pieces les importantes du framework. C'est lui qui vas se charger de choisir une action donnée puis le passer à l'environnement si nécessaire. Dans notre cas on vas utiliser une politique de choix maximum qui vas consiter à prendre le maximum des actions de l'espace d'action.De plus le chef d'orchestre auras la possibilité de choisir l'action suivante avec une probablité ϵ qui vas commencer à $\epsilon = 1$

3. Le réseaux de Neurone

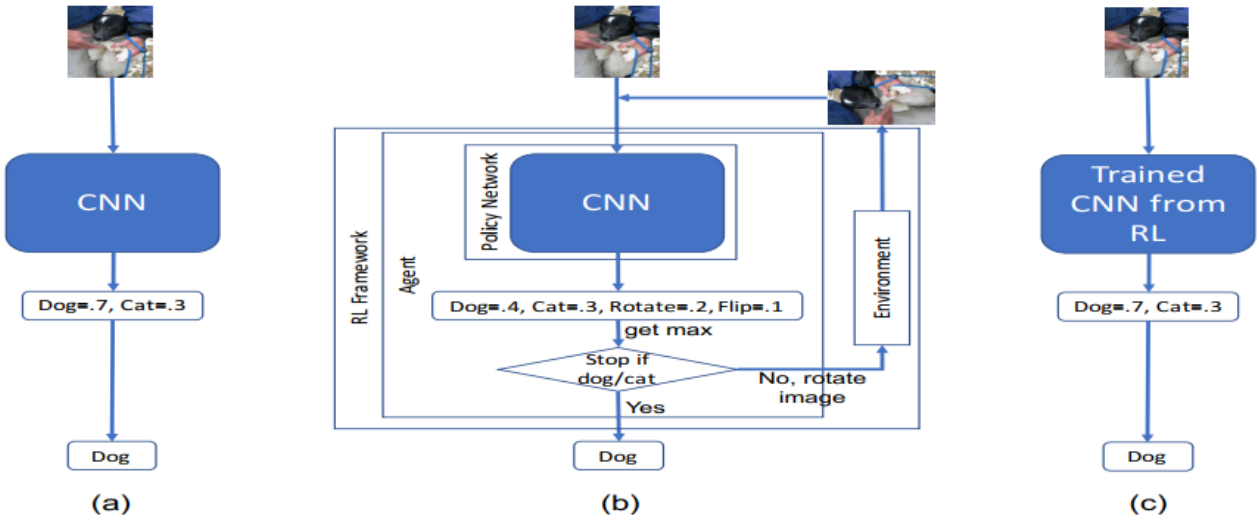
La variante de **DQN** utilisé dans notre étude est un **DQN** nommées Dueling DQN (DDQN). Pour mettre à jour le réseaux de neurone, **l'équations de bellman** seras utilisé. Elle se définit comme suit

$$Q(s,a) = r + \gamma * max_{a'}(Q(s',a'))$$

- $Q(s,a)$: C'est la sortie du neurone pour une action a dans un état s
- r: Est la recompenses retourné par l'environnement
- s': Correspondante à l'état retourné par l'environnement lorque l'action a est appliqué à l'état s
- γ: c'est le paramètre de discount

4. L'environnement

L'environnement est le lieu ou sont appliqué les transformations. Il seras également responsable du calcul des recompenses. Il recoit donc une image et une action du chef d'orchestre puis l'applique. Si l'environnement recoit une action de transformations $T_{actions_i}$, il vérifie si cette chaine d'action a une longueur inférieur à un paramètre m qu'on aura configuré (Pour éviter une saturation de mémoire pour les chaines de transformations très longues). L'action seras uniquement effectué si la longueur est vérifié sinon une autre action seras recherché. Dans le cas ou la longueur est supérieur à m une récompense de zéro est renvoyé puis l'image est restauré. Toutefois à la phase de test, l'action d'arrêt avec la plus grande valeur de Q seras retourné. Dans le cas ou l'environnement recoit une action d'arrêt $S_{actions_j}$, il ne retourne pas d'image au réseaux mais juste une récompenses puis classifie l'image comme une image de classe j. Le systèmes de récompenses est très important dans la convergence de l'algorithme. Une solution simple peut être de retourner $+1$ si après la transformations la classes prédictes de l'image est correcte et -1 sinon. Toutefois avec plusieurs classes de tailles k cette solution n'est pas très efficace. Ainsi pour une prédictions correcte après transformations, l'environnement retourne $k-1$ et -1 sinon.



D. Réseaux de Neurone pour entity Matching

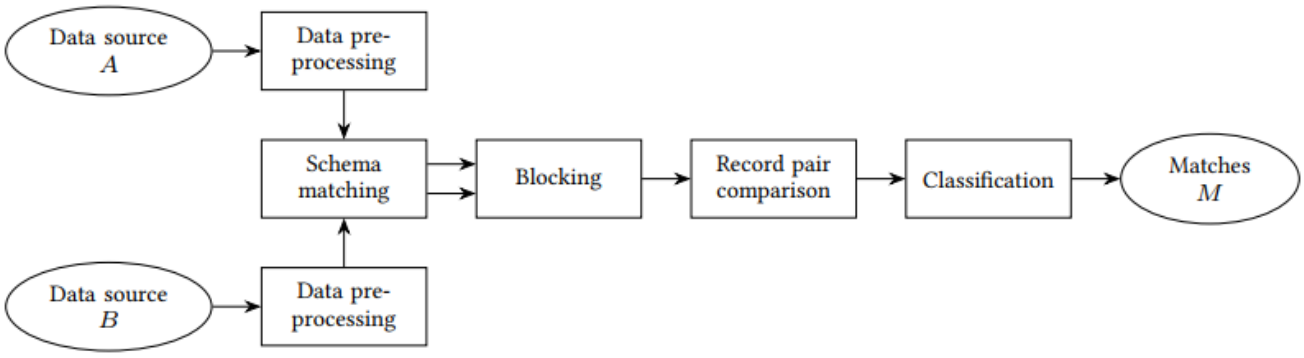
L'**entity matching** peut être considéré comme un secteur de recherche visant à identitifier si deux enrégistrement **A** et **B** réfèrent à la même entité réel. Malgré des décennies de recherches et l'avènment

des nouvelle technologie de machine learning, l'entity mactching reste un problème pour les raisons suivantes:

- **Faible qualité de données**: Les données peuvent contenir des erreurs ou des manques de précisions (par exemple `{firstName: "Machkour Oke", lastName: "Etudiant"})`. De plus elle peuvent être inconsistente, ou suivre un shéma différent en fonction du lieu d'enrégistrement
- **Le grand nombre de match possible (Element similaire)**: Soit un A un dataset de taille n et B de taille n également on peut avoir $n * n$ match soit une complexité de $O(n^2)$. On peut généralement espérer avoir n match. Toutefois comparer toutes les paires est très couteux en calcul.
- **Dépendance envers les connaissances humaines et connaissances domaines**: Bien qu'il soit possible d'automatiser un certains nombre de problème d'entity Matching, un bon nombre de problème ne dispose pas en eux mêmes des informations nécessaires et doivent ainsi faire appel à un expert pour conclure

1 - Définition du problème

Soit A et B deux sources de données de features (caractéristiques) respectifs $(A_1, A_2, ..., A_n)$, $(B_1, B_2, ..., B_m)$. Nous dénoterons $a = (a_1, a_2, ..., a_n) \in A$ et $b = (b_1, b_2, ..., b_m) \in B$ les enrégistrement respectifs de A et B. Le but de **l'entity Matching** seras de trouver le plus grand ensemble $M \subseteq A * B$ telle que pour tout a et b appartenant à M (a, b) refère à la même entité réelle. Le problème de duplication des données est un sous problème de l'entity Matching ou $A = B$



2 - Exemple de problème d'entity Matching

L'entity matching peut être rencontré dans bon nombre de contexte allant de la simple duplication de données à la jointure de données. Quelques problèmes populaires de l'entity matching sont:

- **Résolution des coréférences :** En NLP (Natural language processing), cette tache consiste à trouver dans un texte tous les mots qui refère à la même entité. Cela peut être utile pour effectuer des résumé de texte, des réponses à des questions ou autres
- **Alignement d'entité:** Cette tache correspond à trouver dans deux bases de données les données qui refèrent à la même entité. Cela peut être utile pour détecter un même utilisateurs à travers deux bases de données par exemple

- **Liaison d'entité:** Pour un texte données trouver toutes les mentions d'une entité puis les lier dans une bases de données. Cette opérations peut être vu comme un mélange entre **l'alignement d'entité** et **la résolution de coréférences**
- **Identification de paraphrase:** Pour deux texte données déterminer s'ils ont le même sens. Cela peut être utilisé pour détecter des plagiats par exemple. Toutefois l'entity matching n'est qu'une partie de ce problème

3 - Processus de l'entity matching

Préprocessing des données

Bien que l'entity matching peut être dans l'une des phases de préprocessing, il est important d'effectuer toutes les autres transformations élémentaires aux données. Par exemple s'assurer de la consistance des données, ou parfois extraire les features importante du modèle par exemple

3.1 Correspondance de shéma

Dans cette partie on vas chercher les attributs comparable dans les deux sources de données. Lorsqu'on as une même table ou une structure identique dans les deux tables, la correspondance est trivial. Généralement cette étape seras effectué à la main carson automatisation peut rentrer encore dans un autre types de problèmes

3.2 Blocage

L'ensemble des match potentiels $A * B$ croit de manière quadratique avec la taille des données. Pour éviter d'avoir à comparer toutes les paires possibles, on prend un ensemble $C \subseteq A * B$ de paires candidates. Cela est défini comme une comparaison explicite et moin couteuse qu'une comparaison implicite (qui seras fait par la suite). Elle vas supprimer dans les paires de matches les paires dont le non-matching est évident.Il existe plusieurs techniques pour le faire. On peut par exemple choisir une des colonnes A_i et sa colonne correspondant B_i puis spécifier un seuil de similarité à partir duquel on conserve les pairs.

3.3 Comparaison de pair d'enrégistrement

Dans cette étapes on vas faire une comparaison explicite des des paires $(a, b) \in C$. Cette comparaison vas donner un vecteur de similarité S indiquant la similarité au noveau de chaque attribut entre les deux sources de données

3.4 Classification

L'objectif de cette phase est de déclarer ou non les paires correpondantes. Dans le cas ou $|S| = 1$ on peut fixer un seuil au bout duquel les pairs sont correpondantes ou non. Dans le cas ou $|S| > 1$, des méthodes plus sophistiqu s'impose. Il faut noter qu'on peut obtenir trois type de résultats: **match**, **nonmatch** et incertains. Les paires incertaines auront besoin d'une vérification manuelle

4 - Utilisation des différentes méthodes avec réseaux de Neurone

4.1 Data preprocessing

Les réseaux de neurone comme la plupart des algorithme de machine learning ne fonctionne qu'avec des données numériques. Pour les données textuelle il est donc nécessaire de les convertir en données numérique. Ce processus est appelé l'embedding.

Il existe plusieurs algorithme d'embedding à base de réseaux de neurone comme le **word2vec** qui est un réseaux de neurone que nous n'allions pas détailler ici. L'avantage avec ces réseaux de neurone est qu'ils ont été entrainé sur de large corpus de texte et qu'ils sont disponibles pour plusieurs lagues

4.2 Correspondance de shéma

Pour deux sources de données A et B, les shémas de ces deux sources de donnée peuvent être dans trois cas:

- **Shemas aligné:** Les deux sources de données utilisent le même shéma: $\forall_{i \in (1,2,...,n)}(A_i = B_i)$ et $n = m$.
- **Shémas non aligné** Les deux sources de données ont les mêmes attributs mais dans le mauvais ordre.
- **Shémas incompatible:** Il n'existe pas de correspondance entre les shémas des deux sources

Les deux premiers cas sont généralement simple à résoudre pour le dernier cas, il vas nous falloir des techniques plus elaboré comme celle qui suit:

4.2.1 Comparaison de la réprésentation des attributs

Dans cette méthode on calcule la similarité entre les vecteurs attribut des deux sources (en utilisant la distance cosinus). On peut fixer un seuil pour cette distance au bout duquel on vas déclarer deux attributs comme similaires.

4.2.2 Apprendre les correspondances par attribut dans un clustering

Dans cette méthode on vas tout d'abord effectuer un regroupement non supervisé des attributs. On vas ensuite créer un perceptron multicouche. Ainsi pour un vecteur d'attribut donné le réseau évalue sa similarité avec les catégories de clusters. et ceci est utilisé pour faire correspondre les attributs de la source de données B aux catégories de A.

4.3 Blocage

Pour effectuer le blocage il existe peu d'algorithme de deep learning. On peut citer entre autre les algorithmes DeepER **[3]** et AutoBlock **[3]**. Chaque enrégistrement seras représenté par un vecteur de dimensions égales au nombres de features. Pour trouver les paires candidates de l'ensemble C la distance cosinus seras utilisé. Pour chaque élément $a \in A$ un ensemble d'enrégistrement $B_a =$

$(b_1, b_2, ..., b_i), b_i \in B$ lui seras trouvé. Toutefois avec un grand nombre de caractéristique cette recherche peut être très couteuse, ainsi il vas falloir approximer ces voisins.

4.4 Comparaison de paires

C'est l'élément central de notre opérations. Comme mentionné précédemment le but de cette étape seras de générer un vecteur de similarité S.Pour ce faire on a également bon nombre d'algorithme comme DeepMatcher **[3]**.

Lorsque les attributs sont aligné une comparaison attribut par attribut est réalisé par DeepMatcher. Toutefois si les attriibuts sont non aligné, Deepmatcher fusionne tout les attributs dans une grandes chaines de caractère. Cette méthode est moin efficaces car elle inhibe les informations apportée par la séparation des attributs

Les méthodes générale utilisé pour la comparaison incluent la **différence de vecteur**, le **produit d'Hadamard** (produit élément par élément). **[3]** ou encore le produit cosinus

4.5 Classification

Quelque soit la méthode cette étape est généralement effectué de la même manière, On prend le vecteur similarité S, on le fait rentrer dans un neurone classique puis on fait une classification (**match** ou **nonmatch**)

V - Conclusion

Dans notre étude, nous avions pu voir plusieurs technique de nettoyage de données qui utilise les réseaux de neurones pour leur mise en place.

Ces technique peuvent utiliser un réseaux de Neurone spécialisé pour toute la tache (Imputation et transformation), ou utiliser des réseaux de neurone dans chaque partie de leur processus (Entity Matching).

Toutefois bien que prometteuse, ces méthodes sont confronté à bon nombre de problème comme leur complexité de mise en oeuvre ou leurs temps d'entrainement.

VI - Bibliographie

[1] GUILLAUME SAINT-CIRGUE

Comment fonctionne le Machine Learning ? (2019)

[2] Thirumuruganathan, Saravanan, Tang

Data Curation with Deep Learning [Vision] (2019)

[3] Nils Barlaug, Jon Atle Gulla

Neural Networks for Entity Matching: A Survey (2021)

[4] Hu Pan , Zhiwei Ye, Qiyi He, Chunyan Yan , Jianyu Yuan, Xudong Lai, Jun Su and Ruihan Li

Discrete Missing Data Imputation Using Multilayer Perceptron and Momentum Gradient Descent (2022)

[6] Tran Ngoc Minh , Mathieu Sinn, Hoang Thanh Lam3 , Martin Wistuba

Automated Image Data Preprocessing with Deep Reinforcement Learning (2022)