

## Systèmes Intelligents - IA TD 1 & 2 Python et perceptron




14H rue Pierre de Coubertin  
21000 Dijon  
FRANCE

Tel : +33 380 371 795


23, rue Paul Bert  
92100 Boulogne Billancourt  
FRANCE

Tel : +33 146 032 660

Dans ce premier TD, nous allons apprendre les bases de la programmation Python. Nous utiliserons la version 3.6 du langage, car la version 3 offre des fonctionnalités intéressantes dans le cadre du [parallélisme](#) et du [sucre syntaxique](#) et, en plus de ça, est aussi rapide à exécuter que Python 2.7. Pour rappel, Python 2 n'est plus supporté depuis Janvier 2020.

Pour commencer, vous pouvez cliquer [ici](#) pour une *cheatsheet* Python (et [ici](#) pour télécharger le fichier dont je parle à la fin de la fiche), ou bien vous référer aux divers tutoriels sur internet comme celui de [OpenClassroom](#). Il est évident que tout internet vous est ouvert, et comme l'indique le livre saint de la programmation: "À tout problème correspond un thread [StackOverflow](#)." Pour le code, n'oubliez pas de bien prendre en compte les indications à côté de chaque symbole .


Seul ou en binôme (préférable), vous allez travailler sur un projet d'intelligence artificielle faisant appel aux réseaux de neurones et à l'apprentissage supervisé. Ce projet va s'étaler sur toutes les séances de TD et de TP, il est donc important **de travailler avec quelqu'un de son groupe de TP**. Toutefois, et comme dit plus haut, nous allons avoir besoin de prendre Python en main.

 [PyCharm](#) est un excellent IDE Python qui offre une tonne de fonctionnalités et plugins intéressants, du moins dès lors d'une utilisation poussée du langage. Pour tester des codes simples et fournir des démos figées dans le temps, vous préférerez sans doute [Google Colab](#). De plus, Colab vous offre la possibilité d'utiliser les GPUs et les TPUs de Google ! Très utile quand on ne possède pas le matériel chez soi.

Question 1: Qu'est-ce qu'un fichier CSV ?

Question 2: Qu'est-ce que pip ?

Question 3: Qu'est-ce que le package Numpy ?

 Il n'y a pas d'équivalent à `public static void main()` en Python. En fait, n'importe quel fichier Python peut être exécuté sans qu'il soit considéré comme script principal. Pour pouvoir s'y retrouver on utilise donc [cette technique](#).

Il est temps d'ouvrir votre éditeur Python préféré et de commencer à programmer. Vous avez à votre disposition un fichier CSV contenant des personnages de la série *The Witcher*. Vous devez créer un fichier de script Python principal (que vous n'appellerez évidemment pas `main.py`) et un fichier correspondant à une classe `Character`:

La **classe `Character`** possède un constructeur qui prend 5 arguments: nom, prénom,





Sensing & Predictive AI

[www.yumain.fr](http://www.yumain.fr)

age, profession et boost de moral. Vous devez également lui créer ses propriétés et ses



14H rue Pierre de Coubertin  
21000 Dijon  
FRANCE


Tel : +33 380 371 795

23, rue Paul Bert  
92100 Boulogne Billancourt  
FRANCE

Tel : +33 146 032 660

accesseurs/mutateurs. Cette classe contient également une surcharge de la méthode `__repr__()` ; c'est l'équivalent du `toString()` en Java.

Le **script principal** doit ouvrir le fichier CSV fourni et créer un objet par personnage. Pour répondre à votre question: Non, `name,first name,age,profession,morale value` n'est bien évidemment pas un personnage et vous devez donc sauter cette ligne. Tous les personnages doivent être ajoutés à une liste nommée `characters_list`.

 Il existe en Python le package CSV. Pour importer un package, il suffit d'écrire en haut de votre fichier

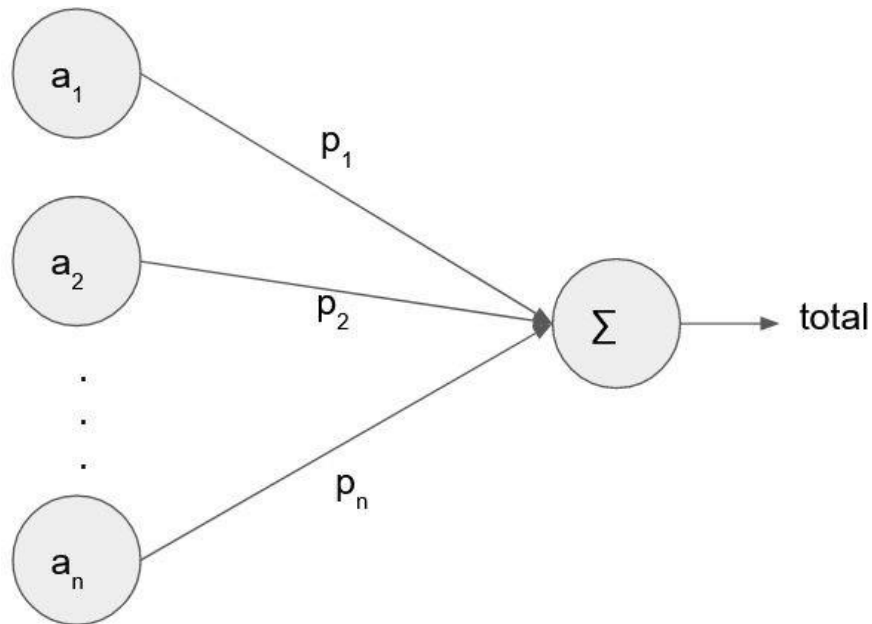
```
import nom_du_package
```

Créez ensuite un nouveau fichier qui contiendra une **classe Army**. Une armée est constituée d'un.e chef.fe et d'une valeur de moral de base. Écrivez à nouveau les accesseurs/mutateurs et la méthode `__repr__()`. Ajoutez lui également une méthode `get_total_morale()`. Celle-ci doit renvoyer la valeur de moral de base de l'armée multipliée par le coefficient de boost moral de son.a chef.fe.

De retour dans le **script principal**, faites en sorte de créer une armée par personnage, et d'assigner à chaque personnage son armée. La valeur de moral d'une armée est un flottant aléatoire entre 20 et 100. Calculez la valeur totale de moral de toutes les armées.

Vous vous êtes normalement rendus compte que nous n'avons fait que multiplier une valeur par un coefficient et fait une somme tout bête à la fin. Ce qui pourrait donc être représenté par ce schéma assez suspect et potentiellement vu lors d'un CM d'IA.





avec  $a_i$  la valeur de moral et  $p_i$  le coef. du personnage



Pas de droit d'installation ? Pas de problème ! Pour installer un package Python vous pouvez tout simplement exécuter la commande suivante dans votre terminal:

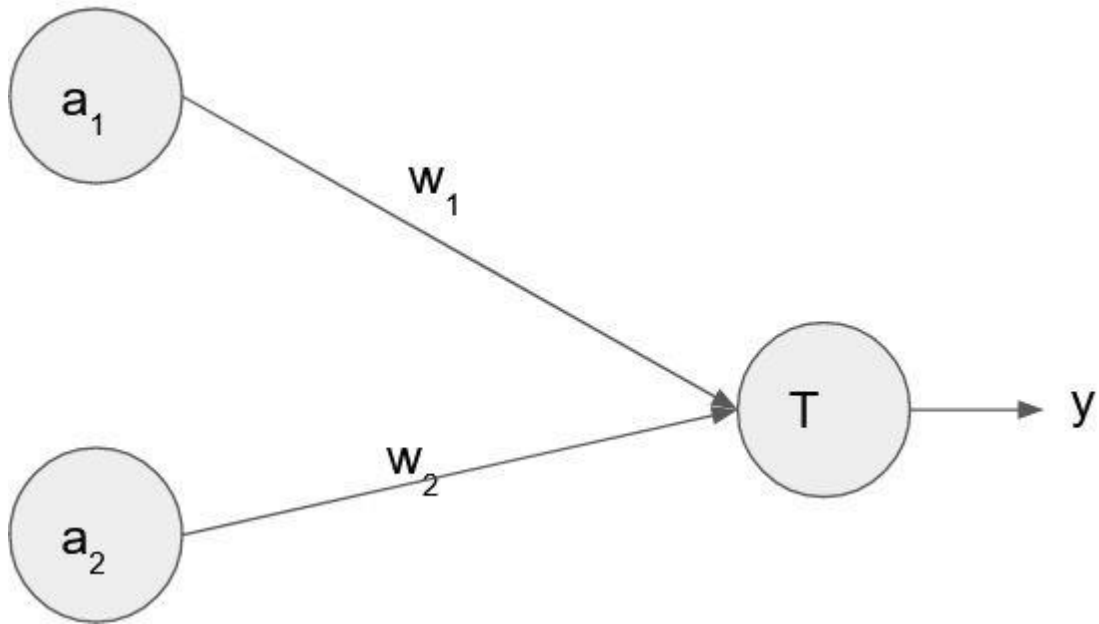
`pip install package_name --user`

**Vous pouvez également installer des package sur Colab !**

Notre système entièrement réalisé en programmation orientée objet pourrait être réalisé bien plus rapidement en utilisant simplement des tableaux Numpy. Créez un tableau Numpy de 5 flottants aléatoires correspond à la valeur de moral des armées. Créez un tableau Numpy de 5 flottants correspondant au boost moral des personnages. Calculez la somme des valeurs de moral total des armées (comme ce qui a été fait précédemment). Tout doit tenir en 3 lignes de code **au maximum**. Pensez à utiliser les fonctions offertes par Numpy.

Passons désormais à des choses un peu plus intéressantes. Nous allons réaliser notre premier apprentissage avec un perceptron. Nous allons faire en sorte que notre petit réseau de neurones apprenne la fonction AND. Mais avant de se faire, nous allons vérifier que ce modèle ci-dessous peut fonctionner grâce à la visualisation de sa surface d'erreur.





avec  $a$  les inputs,  $w$  les poids,  $T$  la fonction d'activation et  $y$  la sortie  
Ici,  $T$  est la fonction qui à  $x$  associe 0 si  $x \leq 0$ , sinon 1.

- Créez une liste à 2 dimensions contenant les inputs possibles pour la fonction AND.  $[[0, 0], [0, 1], [1, 0], [1, 1]]$
- Créez une liste à 1 dimension contenant les sorties attendues pour chaque paire d'inputs.  $[0, 0, 0, 1]$
- A l'aide de deux boucles for, faites varier  $w_1$  de -5 à 5 et  $w_2$  de -5 à 5. Calculez la prédiction du réseau pour les valeurs de poids que vous avez et déterminez la valeur d'erreur associée. Ajoutez cette valeur d'erreur dans une liste à 2 dimensions. Avec l'aide de matplotlib, affichez une image représentant la surface d'erreur.

La valeur d'erreur est donnée par la relation suivante:

$$E = \frac{1}{2} \sum_{\forall i} (y_i - t_i)^2$$

Ou bien, dans le cas où nous n'avons qu'une seule sortie (comme ici):

$$E = \frac{1}{2} (y - t)^2$$



Cette partie peut être complexe à comprendre, alors voici un exemple:

Je sais que mes poids peuvent chacun aller de -5 à 5, donc je crée une liste à 2 dimensions de taille 10x10. Ainsi la valeur d'erreur lorsque  $w_1$  vaut -5 et  $w_2$  vaut -5 sera stocké dans  $[0][0]$ , la valeur d'erreur lorsque  $w_1$  vaut -5 et  $w_2$  vaut -4 sera stocké dans  $[0][1]$ , la valeur d'erreur lorsque  $w_1$  vaut -5 et  $w_2$  vaut -3 sera stocké dans  $[0][2]$ , etc.

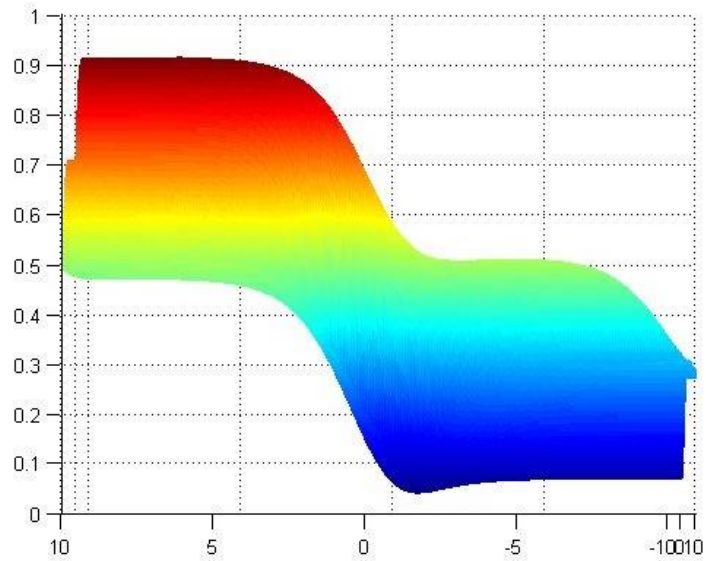
Je vais à chaque fois tenter de faire une prédiction avec chacun de mes inputs.

Prenons par exemple le cas où  $w_1$  vaut -5 et  $w_2$  vaut -5:

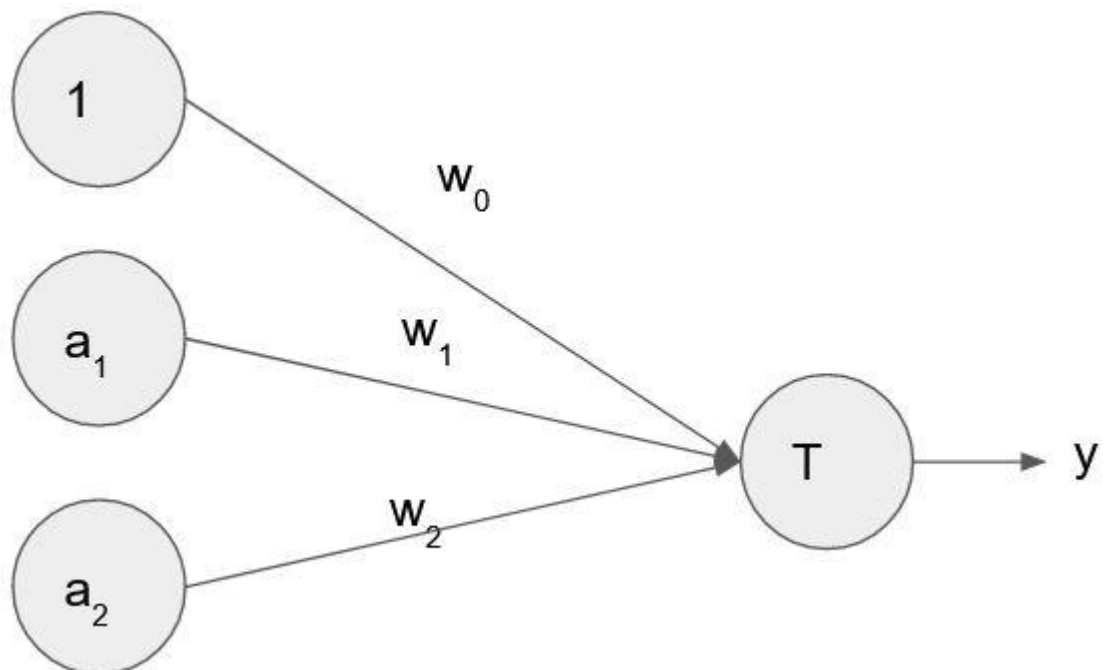
- Notre perceptron fait toujours le calcul suivant  $\rightarrow w_1 \cdot a_1 + w_2 \cdot a_2$
- On substitue  $a_1$  et  $a_2$  par le premier input, soit 0,0
- $w_1 \cdot a_1 + w_2 \cdot a_2$  devient  $-5 \cdot 0 + -5 \cdot 0$  ce qui est égal à 0. Comme  $0 \leq 0$ , le neurone ne s'active pas.
- La sortie attendue, appelée  $t$ , vaut 0
- La valeur d'erreur sur cette prédiction vaut donc  $\frac{1}{2} \cdot (0-0)^2 = 0$
- La valeur d'erreur totale sur ces poids vaut 0.
- On substitue  $a_1$  et  $a_2$  par le deuxième input, soit 0,1
- $w_1 \cdot a_1 + w_2 \cdot a_2$  devient  $-5 \cdot 0 + -5 \cdot 1$  ce qui est égal à -5. Comme  $-5 \leq 0$ , le neurone ne s'active pas.
- La sortie attendue, appelée  $t$ , vaut 0
- La valeur d'erreur sur cette prédiction vaut donc  $\frac{1}{2} \cdot (0-0)^2 = 0$
- La valeur d'erreur totale pour ces poids vaut  $0+0=0$
- On substitue  $a_1$  et  $a_2$  par le troisième input, soit 1,0
- $w_1 \cdot a_1 + w_2 \cdot a_2$  devient  $-5 \cdot 1 + -5 \cdot 0$  ce qui est égal à -5. Comme  $-5 \leq 0$ , le neurone ne s'active pas.
- La sortie attendue, appelée  $t$ , vaut 0
- La valeur d'erreur sur cette prédiction vaut donc  $\frac{1}{2} \cdot (0-0)^2 = 0$
- La valeur d'erreur totale pour ces poids vaut  $0+0+0=0$
- On substitue  $a_1$  et  $a_2$  par le dernier input, soit 1,1
- $w_1 \cdot a_1 + w_2 \cdot a_2$  devient  $-5 \cdot 1 + -5 \cdot 1$  ce qui est égal à -10. Comme  $-10 \leq 0$ , le neurone ne s'active pas.
- La sortie attendue, appelée  $t$ , vaut 1
- La valeur d'erreur sur cette prédiction vaut donc  $\frac{1}{2} \cdot (0-1)^2 = 0.5$
- La valeur d'erreur totale pour ces poids vaut  $0+0+0+0.5=0.5$
- Je stocke la valeur 0.5 à l'index  $[0][0]$  de ma liste

Nous allons introduire la notion de **biais**. Un biais est une valeur qui vient s'ajouter dans notre réseau pour aider celui-ci à apprendre. Le biais pousse les résultats à aller dans un sens. Imaginons la surface d'erreur suivante pour un tout autre problème:





On voit que la valeur d'erreur n'atteint jamais 0, mais au plus bas 0.05. Ca veut dire que si mon réseau pouvait obtenir une valeur constante supplémentaire dans son calcul, je serais capable d'avoir le bon résultat d'apprentissage ! Dans mon perceptron, cela se représente de la manière suivante:





Bien évidemment, ce poids  $w_0$  doit aussi être corrigé lors de l'apprentissage.

Créez désormais 2 fichiers. Un script principal et un autre contenant une classe Perceptron. Le constructeur du perceptron prend le nombre d'inputs du perceptron, le nombre d'époques (càd le nombre de boucle d'apprentissage) et un learning rate. Le perceptron possède des poids initialisés à 0. Il possède également un biais de valeur 1 et dont le poids vaut 0 (voir le schéma ci-dessus). Faites une méthode *predict* qui prend en argument une paire d'inputs et renvoie 1 ou 0 selon la [règle du perceptron](#). Faites une méthode *train* qui prend en entrée la liste des inputs possibles et les sorties attendues pour chaque paire d'inputs.

La méthode train doit, en son sein:

- Autant de fois qu'il y a d'époques:
  - Pour chaque paire d'inputs:
    - Faire une prédiction
    - Au vu de la prédiction, corriger la valeur des poids via la [loi de Widrow-Hoff](#)

Votre script principal doit créer un perceptron capable de résoudre la fonction AND. Enregistrez la valeur des poids trouvés dans un fichier CSV.



Tu as déjà dû l'utiliser plus haut, mais la [fonction open](#) de Python permet de nombreuses choses. D'ailleurs au lieu de faire

```
f = open()
#do something here
f.close()
```

Tu pourrais aller beaucoup plus vite en faisant

```
with open() as f:
    #do something here
#file closes by itself when indentation is over
```

