AI Workshop

Bonjour et bienvenue à votre premier workshop IA organisé par PoC !

Ce workshop sera organisé de la façon suivante: nous vous définirons les différences entre IA, machine learning et deep learning. Puis nous vous énumérerons quelques cas d'applications dans la vie réelle. Ensuite, nous vous présenterons les différents types d'apprentissage tels que supervisés, non supervisés et par renforcement. Enfin nous nous concentrerons particulièrement sur un problème de classification binaire à travers un exemple simple de machine learning.

Pour résumer:

I. IA, Machine Learning & Deep Learning
   a) IA
   b) Machine Learning
   c) Deep Learning
   d) Exemples

II. Types d'apprentissage
   a) Apprentissage supervisé
   b) Apprentissage non supervisé
   c) Apprentissage par renforcement

III. Reconnaître un chat

I. IA, Machine Learning & Deep Learning

a) L'IA

L'intelligence artificielle est un programme capable d'effectuer des tâches humaines. Admettons que nous voulions savoir si une personne est en surpoids ou pas. D'une part nous pouvons récupérer les données concernant son poids, sa taille et y appliquons une formule mathématique qui va nous retourner un score. D'autre part nous avons défini un seuil. Si ce score est inférieur à ce seuil, alors cette personne n'est pas en surpoids. Sinon, elle l'est. Vous l'avez compris, une série de if...else… peut être considérée comme une intelligence artificielle. Ce n'est bien entendu pas ce que nous allons faire pendant nos workshops.

b) Le Machine Learning

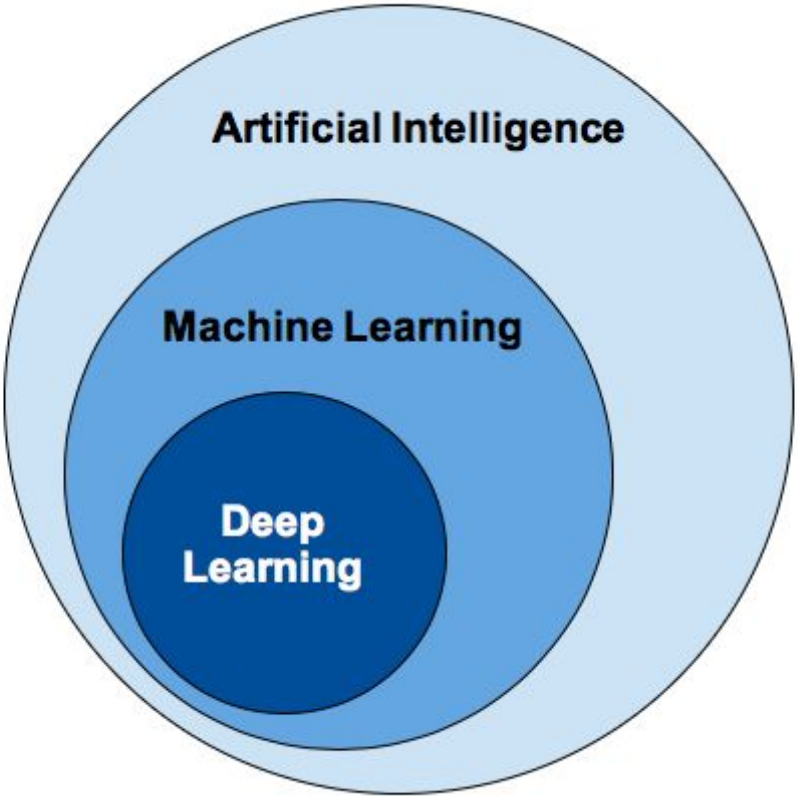
Le Machine Learning est une sous-branche de l'IA. Autrement dit, tout ce qui est Machine Learning est IA mais l'IA ne se limite pas au Machine Learning. C'est un programme capable d'apprendre et de résoudre un problème sans avoir été explicitement programmée. Ceci est possible grâce à des algorithmes dont nous vous donnons quelques exemples:

- régressions (linéaire, logistique, etc.)
- clustering
- arbres de décision
- Naïve Bayes
- etc.

Derrière ces algorithmes se cachent des formules mathématiques.

c) Le Deep Learning

Le Deep Learning est une sous-branche du Machine Learning. Son fonctionnement est issu de celui du cerveau humain et ses neurones et l'information est transmise de neurones à neurones, d'un poids de départ à un point d'arrivée. Nous vous donnerons plus de détails plus tard.



d) Exemples

Voici quelques exemples d'applications:

- Reconnaître un chat sur une image
- Détecter le sentiment d'un avis pour un film
- Classifier des articles selon leur thème
- Prédire le prix d'un appartement
- Détecter des spams
- Deepfake
- Retranscrire des paroles en texte
- Détecter un cancer
- Chatbot
- Smartspeakers: Google Home / Alexa / etc.
- Recommandation de films: Netflix
- Reconnaître des objets
- Battre le champion du monde de go
- Etc.

II. Types d'apprentissage

Lorsque vous voulez faire apprendre une tâche à une machine, vous devez lui fournir des données en entrée. Votre modèle va permettre de vous sortir une ou des données de sortie.
Par exemple, admettons que vous souhaitez prédire le prix d'une maison, les données en entrées peuvent être la superficie de la maison, le nombre de chambres et le quartier. En sortie, ce sera le prix.
Autre exemple, vous souhaitez savoir si une image contient un chat. Les données en entrée peuvent être les pixels d'une image, et la donnée de sortie sera "oui" ou "non".

Il existe maintenant plusieurs manières de résoudre ces problèmes. On distingue généralement 3 types d'apprentissage: l'apprentissage supervisé, non supervisé et par renforcement.

a) L'apprentissage supervisé

L'apprentissage supervisé se dit "supervisé" car vous indiquez à votre modèle le résultat attendu. Dans le cas de notre exemple de chat, lors de la phase d'apprentissage, vous donnerez au modèle non seulement des images de chat, mais vous indiquerez également à l'algorithme quelles images comprennent des chats et celles qui n'en contiennent pas. Le modèle est donc capable d'adapter ses poids en fonction du résultat attendu (ce n'est pas grave pour l'instant si vous ne comprenez pas tout).

b) L'apprentissage non supervisé

Comme vous vous en doutez, contrairement à l'apprentissage supervisé, un modèle d'apprentissage non supervisé n'a pas besoin que vous lui spécifiez le résultat attendu. Ce sera au modèle d'apprendre seul.

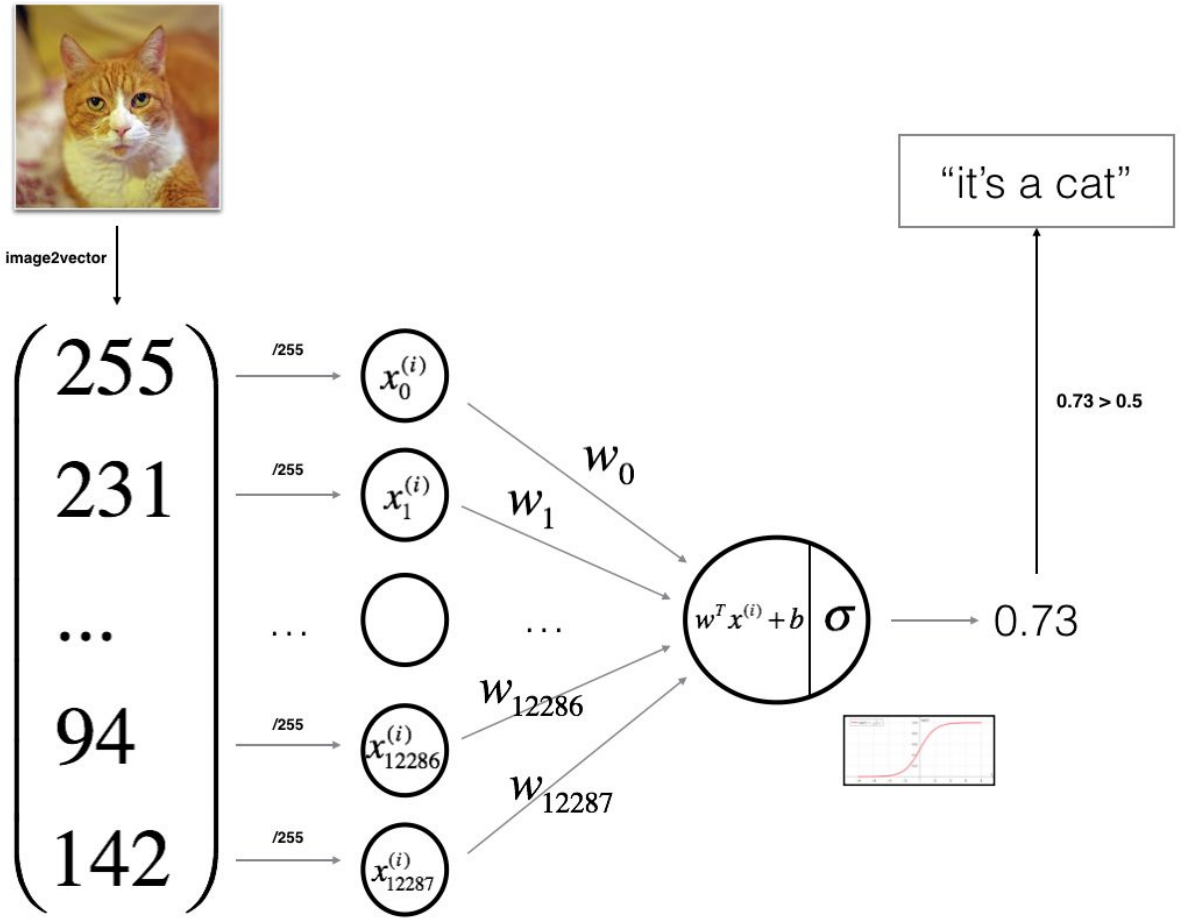
c) L'apprentissage par renforcement

Il s'agit d'un apprentissage différent des deux autres qui fonctionne sur un système de récompenses/punitions. Prenons l'exemple des échecs: le but de l'IA sera de gagner à toutes les parties. Pour ce faire, pendant l'entraînement du modèle, l'IA recevra une récompense à chaque partie gagnée et une punition à chaque partie perdue. Ainsi, après plusieurs milliers voire millions de parties jouées, l'IA sera capable de gagner (ou du moins d'avoir les chances maximum) à toutes les parties.

III. Reconnaître un chat

Maintenant qu'on a fini l'introduction, on va pouvoir passer aux choses sérieuses.

On va traiter un problème de classification binaire: reconnaître un chat sur un image.
Si l'image contient un chat alors le modèle devra prédire 'cat', sinon 'not cat'. Pour cela, on va entraîner notre modèle de façon supervisée en lui fournissant un jeu de données de 259 images de dimension 64*64. En effet, à chaque fois que vous avez un jeu de données et que vous voulez appliquer un modèle de machine learning dessus, vous devez d'abord créer votre modèle et l'entraîner sur vos données. Mais vous devez aussi vérifier que votre modèle créé fonctionne. La première étape va être de séparer notre jeu de données en 2 jeux: un qui servira à l'entraînement (train) et un autre à la vérification (test). Le ratio train/test dépend de la quantité de données que vous avez, en général 80/20 est un bon ratio. Dans notre cas, nous prendrons 209 en train et 50 en test.



Cette image représente notre modèle de machine learning et prend en entrée tous les pixels (de 0 à 255) de cette image. Il retourne en sortie un score compris entre 0 et 1 permettant de savoir s'il s'agit d'un chat.

Comment le modèle parvient-il à faire tout cela ?

- Les poids

Grâce aux “flèches” w_0 à w_{12287} qui représentent des poids. Reprenons les bases du collège: sur un graphique, une fonction affine est représenté par une ligne droite et s’écrit de la forme **f(x) = ax + b**. a et b sont des poids car ce sont eux qui définissent le degré de pente et la hauteur de la droite. Dans notre cas et dans tout modèle de classification, le but sera de faire apprendre ces poids à notre modèle.

- Les fonctions de pré-activation et activation

Les neurones x_0, x_1 à x_{12287} représentent donc les features en entrée de notre modèle.
Au début, les poids (représentés par w) sont initialisés soit à 0 soit de façon aléatoire.

Vous voyez ensuite que le neurone suivant a été divisé en 2 parties:

- d’une part on a $w^T x^{(i)} + b$ qui représente la fonction de pré-activation
- d’autre part σ (sigma, représentant la fonction sigmoïde, on vous expliquera plus loin)

Cela représente les étapes de calcul d’un neurone:

- l’étape de pré-activation, obligatoire, représentée par la formule suivante:

  $z^{(i)} = w^T x^{(i)} + b$ avec

  z = pré-activation, w^T = transposée de la matrice des poids,

  $x^{(i)}$ = features en entrée et b = biais
- la fonction d’activation, dans notre cas la fonction sigmoïde que nous vous expliquerons plus tard mais ça peut être d’autres fonctions.

Enfin, 0.73 représente le résultat de la fonction sigmoïde qui va nous permettre de valider si l’image donnée en entrée correspond à un chat ou non.

- Les fonctions de loss et de coût

Arrivé à cette étape, on peut déjà vérifier si notre modèle est pertinent. Pour cela, on va appliquer sur chaque neurone la fonction de loss représenté par la formule suivante:

$$L(a^{(i)}, y^{(i)}) = -y^{(i)} \log(a^{(i)}) - (1-y^{(i)}) \log(1-a^{(i)})$$

où $a^{(i)}$ = activation et $y^{(i)}$ = la vraie valeur

Ensuite, on appliquera la fonction de coût qui n’est autre que la moyenne des loss de tous les neurones. Notre but sera donc de minimiser ce coût, et c’est là qu’intervient …

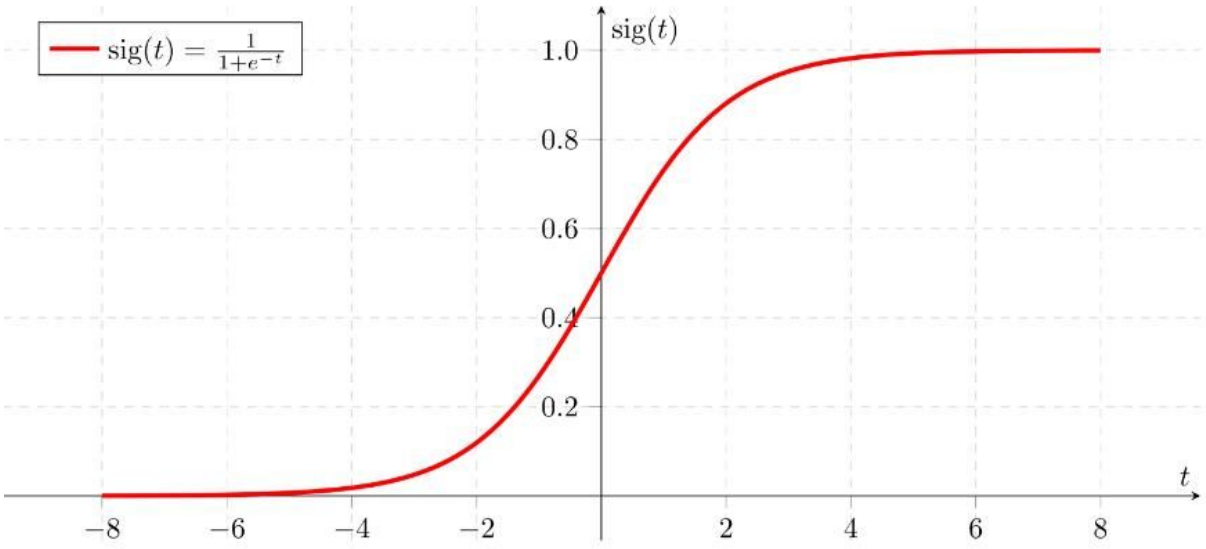
- La descente de gradient

La descente de gradient est un algorithme permettant de minimiser le coût. Voici un très bon lien expliquant bien cet algorithme: https://www.charlesbordet.com/fr/gradient-descent/#
Il va vous permettre de calculer les gradients des poids et du biais et de pouvoir les update grâce à la formule suivante: $\theta = \theta - \alpha d\theta$ où α correspond au learning rate (le pas d’apprentissage).

Maintenant qu'on est allés assez en profondeur sur le sujet. On va prendre un peu de recul et combiner toutes ces notions ensemble.
L'apprentissage se fait par une succession de forward propagation, calcul de coût, et backward propagation.
La forward propagation va permettre de calculer les poids de nos neurones et la backward propagation va permettre de les corriger, grâce à la fonction de coût.
Sur notre schéma, l'étape de forward propagation se suit de gauche à droite et comprend les fonctions de pré-activation et activation. Ensuite se tient l'étape de calcul des loss et du coût, et enfin l'étape de back-propagation repart de droite vers la gauche et comprend le calcul des gradients et l'update des poids.
Rappelons que la descente de gradient n'est qu'un algorithme parmi d'autres.

La fonction sigmoïde

On avait dit qu'on parlerait de la fonction sigmoïde ! Voici sa courbe:



Comme vous pouvez le voir sur cette image, la fonction sigmoïde est comprise entre 0 et 1. Cette fonction est très utilisée dans le cas d'une classification binaire (comme notre exemple) et est généralement suivie par l'étape suivante: tout ce qui est supérieur à 0.5 sera considéré comme 1 et inférieur à 0.5 à 0.

Dans le cas de notre exemple: on a le neurone de sortie égal à 0.73. Comme il est supérieur à 0.5, on le considère comme 1 (cat). Cela signifie que l'image donnée en entrée correspond bien à un chat.

Il existe d'autres fonctions d'activation que la fonction sigmoïde (tanh, relu, etc.)

Evaluation de notre modèle

On peut évaluer notre modèle à travers différentes métriques. Prenons la métrique de précision par exemple, qu'on nomme "accurary" en anglais, et qui correspond à un pourcentage.
On peut l'appliquer sur nos deux jeux de données (train et test).
Plus l'accuracy est élevé, plus le modèle est précis.
Si les accuracy de train et de test ont des valeurs proches, cela signifie que notre modèle généralise bien le problème, bingo.
Cependant si l'accuracy de train est élevé alors que celui de test est faible, cela signifie que notre modèle est sur-entraîné. Il a trop appris sur les images de train et ne serait capable de reconnaître que celles-ci, mais pas celles qu'il n'a jamais rencontrées. On appelle cela l'over-fitting.

Prochaine étape:

Ok, maintenant qu'on a pu voir pas mal de théorie, on va passer à l'étape pratique. Vous aurez besoin d'installer Jupyter Notebook (https://jupyter.org/install). Le mieux est d'installer directement la distribution Anaconda comme ils vous le conseillent sur le site, mais si vous êtes déjà à l'aise avec Python et tous ses packages, vous pouvez tout installer manuellement. Pour faire simple, Jupyter Notebook va vous permettre d'exécuter du code pas à pas et vous n'aurez pas besoin de lancer tout le script comme c'est le cas souvent en Python.

Après avoir installé Jupyter Notebook, clonez le repo à l'adresse suivante:

https://github.com/vincent-lemesle/workshop_IA_2019

Ce repo contient les différents notebooks que vous aurez besoin de compléter. Veuillez commencer par le 1 sur les basiques de Python et Numpy avant d'attaquer le 2. Nous sommes à votre disposition pour toute aide.