CLAVIER Olivier
MAZEAU Clément
GROSSI Julia

Compte rendu du projet d'ADIMAKER en informatique

Sujet : Développement d'une intelligence artificielle

Objectif : L'objectif du projet est de créer une application qui reconnait en temps réel des nombres que vous présenterez avec vos mains devant votre webcam. La reconnaissance d'un nombre particulier devra lancer l'exécution d'une application de votre choix (par exemple : Représenter le chiffre 4 devra exécuter un navigateur web ou un logiciel de votre choix, sur votre ordinateur)

Dans le cadre d'ADIMAKER en Informatique nous devions créer une intelligence artificielle capable de détecter une main et de l'analyser.

La première étape a été d'effectuer des recherches théoriques sur le fonctionnement d'une intelligence artificielle.

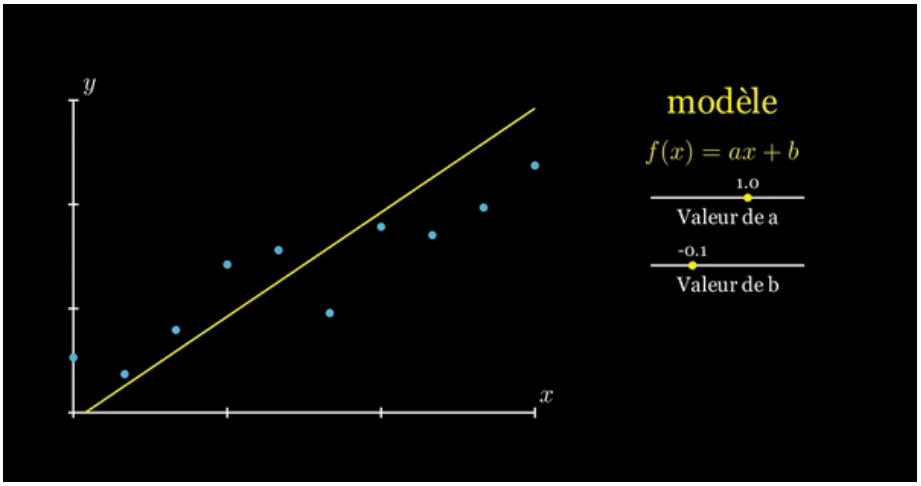
Quand on parle d'intelligence artificielle on parle de machine Learning, de réseaux de neurones et également de Deep Learning.

C'est quoi le machine Learning ?

Le machine Learning est un domaine de l'intelligence artificielle qui consiste à programmer une machine pour que celle-ci apprenne à réaliser des tâches en étudiant des exemples de ces dernières. D'un point de vue mathématique, ces exemples sont représentés par des données que la machine utilise pour développer un modèle.

Ici on va utiliser l'exemple de la fonction type ***f(x) = ax + b***

La machine va chercher la meilleure valeur d'**a** et de **b** qui donnera le meilleur modèle qui s'ajuste le mieux aux données.



Source : https://www.youtube.com/watch?v=XUFLq6dKQok&list=PLO_fdPEVlfKoanjvTJbIbd9V5d9Pzp8Rw&index=1

Pour cela on programme dans la machine un algorithme d'optimisation qui va tester différentes valeurs pour **a** et pour **b** afin d'obtenir la combinaison qui minimise la distance entre le modèle et les points.

Dans les grandes lignes, le machine Learning consiste à développer un modèle en se servant d'un algorithme d'optimisation pour minimiser les erreurs entre les modèles de nos données.

Ils existent de nombreux modèles d'apprentissage venant avec leur propre algorithme d'optimisation

- Modèle linaire -> Descente de gradients
- Arbres de décision -> Algorithme CART
- Support Vectors machines -> Marge maximum

C'est quoi le Deep Learning ?

Le Deep Learning est un domaine du machine Learning dans lequel, au lieu de développer un modèle précédemment cité, on développe à la place ce qu'on appelle un réseau des neurones artificiels.

Le principe reste principalement le même, on fournit à la machine des données et elle utilise un algorithme d'optimisation pour ajuster le modèle à ces données. Mais ici le modèle n'est pas une simple fonction du type ***f(x) = ax + b*** mais plutôt un réseau de fonctions connectées les unes aux autres, un ***réseau de neurones.***

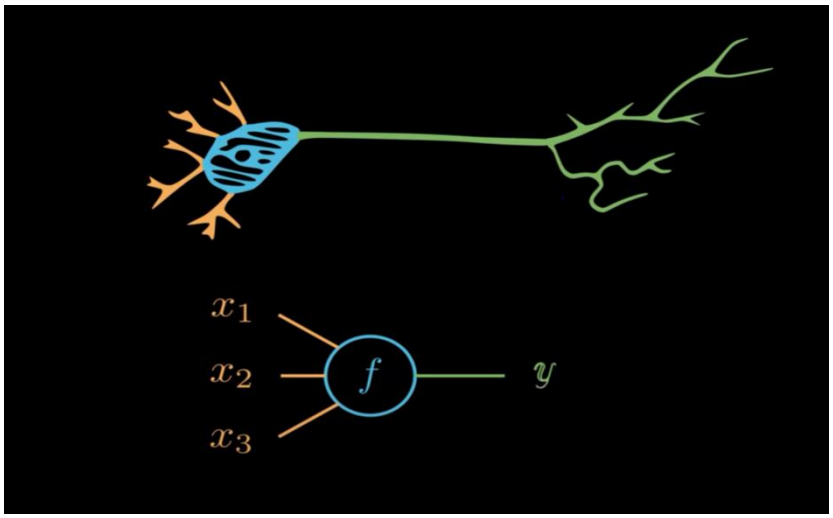
Plus ces réseaux sont profonds, c'est-à-dire plus ils contiennent de fonctions à l'intérieur, plus la machine est capable d'apprendre à effectuer des tâches complexes comme reconnaître des objets, identifier une personne sur une photo ou même conduire une voiture.

Les premiers réseaux de neurone on était inventé en 1943 par deux mathématiciens et neuroscientifiques du nom de Warren McCulloh et Walter Pitts. Ils expliquent comment ils ont programmé des neurones artificiels en s'inspirant du fonctionnement des neurones biologiques.

Pour rappel, en biologie, les neurones sont des cellules excitables connectées les unes aux autres, et ayant pour rôle de transmettre des informations dans notre système nerveux. Chaque neurone est composé de ***plusieurs dendrites***, d'un corps cellulaire, et ***d'un axone.***

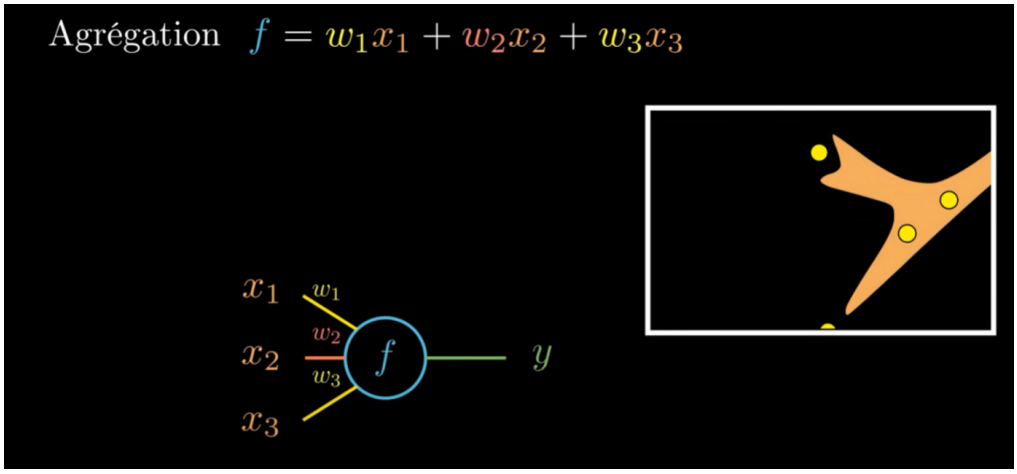
Les dendrites sont en quelque sorte les portes d'entrée d'un neurone, c'est à cet endroit, au niveau de la synapse, que le neurone la synapse, que le neurone reçoit des signaux lui provenant des neurones qui le précèdent. Ces signaux peuvent être de type excitateur ou à l'inverse inhibiteur (un peu comme si nous avions des signaux qui valent + 1 et d'autres qui valent -1). Lorsque la somme de ces signaux dépasse un certain seuil, le neurone s'active et produit alors un signal électrique, ce signale circule le long de l'axone en direction des terminaisons pour être envoyé à son tour vers d'autres neurones de notre système nerveux (neurones qui fonctionnent de la même manière).

Warren McCulloh et Walter Pitts ont essayer de modéliser ce fonctionnement en considérant qu'un neurone pouvait être représenté par une fonction de transfert qui prend en entrée des signaux X et qui retourne une sortie Y :



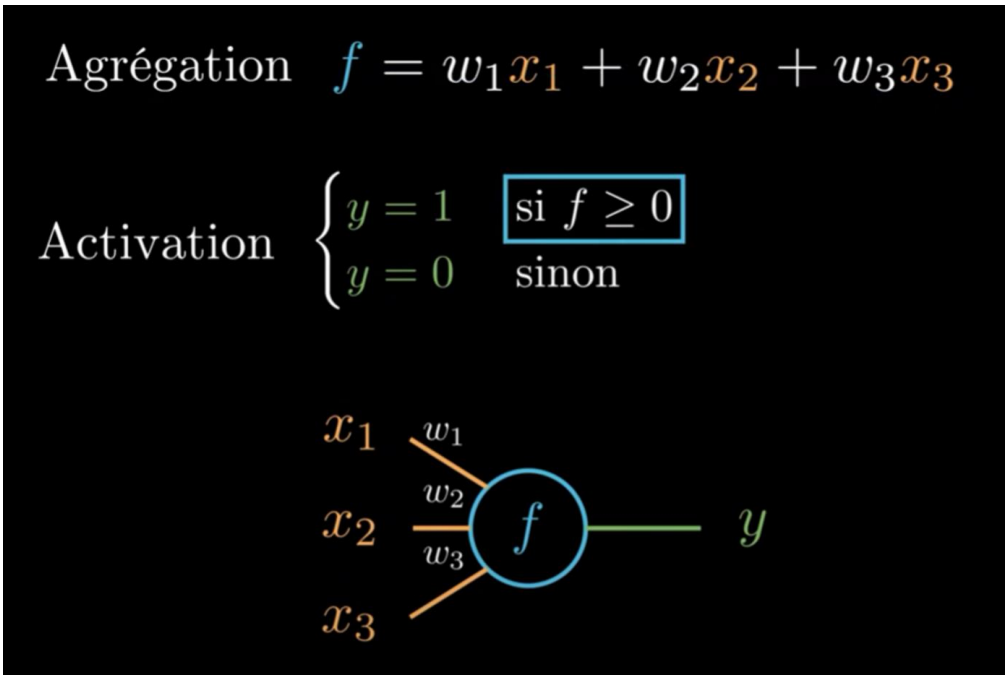
Source : https://www.youtube.com/watch?v=XUFLq6dKQok&list=PLO_fdPEVlfKoanjvTJbIbd9V5d9Pzp8Rw&index=1

A l'intérieur de cette fonction on trouve 2 grandes étapes. La première, c'est une étape d'agrégation. On fait la somme de toutes les entrées du neurone en multipliant au passage chaque entrée par un coefficient w, ce coefficient représente l'activité synaptique, si le signal est excitateur w vaut +1, si le signal est inhibiteur w vaut -1.



Source : https://www.youtube.com/watch?v=XUFLq6dKQok&list=PLO_fdPEVlfKoanjvTJbIbd9V5d9Pzp8Rw&index=1

Une fois cette étape réalisée, on passe à la phase d'activation. On regarde le résultat du calcul effectué précédemment et si celui-ci dépasse un certain seuil, en général 0, alors un neurone s'activé et retourne une sortie y = 1. Sinon, il reste à 0.

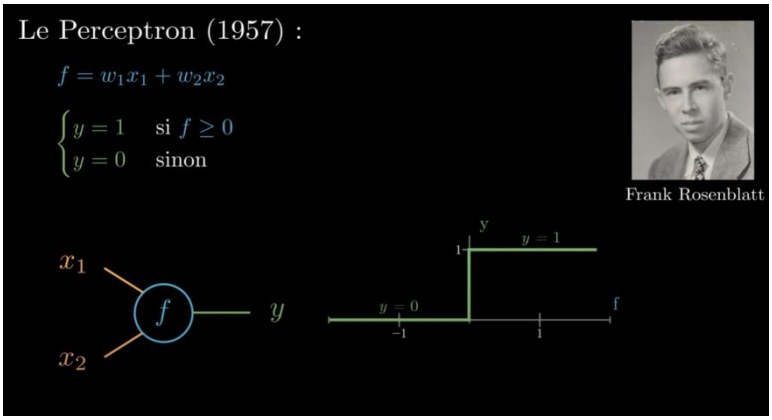


Source : https://www.youtube.com/watch?v=XUFLq6dKQok&list=PLO_fdPEVlfKoanjvTJbIbd9V5d9Pzp8Rw&index=1

Voici les premiers neurones artificiels (Threshold Logic Unit). Ce nom vient du fait qu'à l'origine, leur modèle n'était conçu que pour traiter des entrées logiques qui valent soit 0 à 1. Ils ont pu démonter avec ce modèle qu'il était possible de reproduire certains fonctions logique, telles que la porte AND et la porte OR.

Ils ont également démontré qu'en connectant plusieurs de ses fonctions les unes aux autres, un peu à la manière des neurones de notre cerveau, alors il serait possible de résoudre n'importe quel problème de logique booléenne. Or il contient de nombreuses failles notamment le fait qu'il ne dispose pas d'algorithme d'apprentissage, il faut donc trouver nous même les valeurs des paramètres w si l'on désire s'en servir pour des applications du modèle réel.

15 ans plus tard, en 1950, *Frank Rosenblatt* invente ce qu'on appelle un perceptron. Le perceptron ressemble de très près à celui que nous avons étudier.

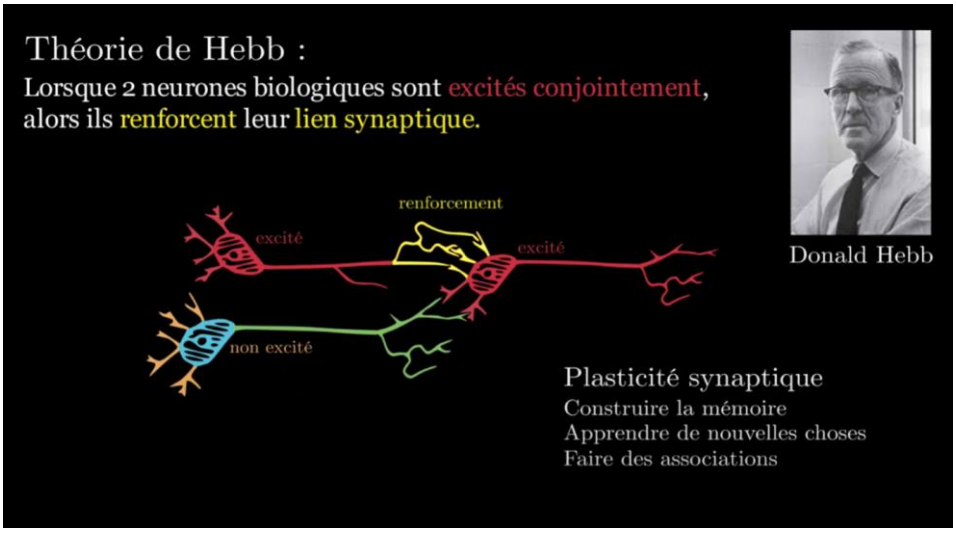


Source : https://www.youtube.com/watch?v=XUFLq6dKQok&list=PLO_fdPEVlfKoanjvTJbIbd9V5d9Pzp8Rw&index=1

Il s'agit d'un neurone artificiel, qui s'active lorsque la somme pondérée de ses entrées dépasse un certain seuil, en général 0. Mais avec ça, le perceptron dispose également d'un algorithme d'apprentissage lui permettant de trouver les valeurs de ses paramètres w afin d'obtenir les sorties y qui nous conviennent.

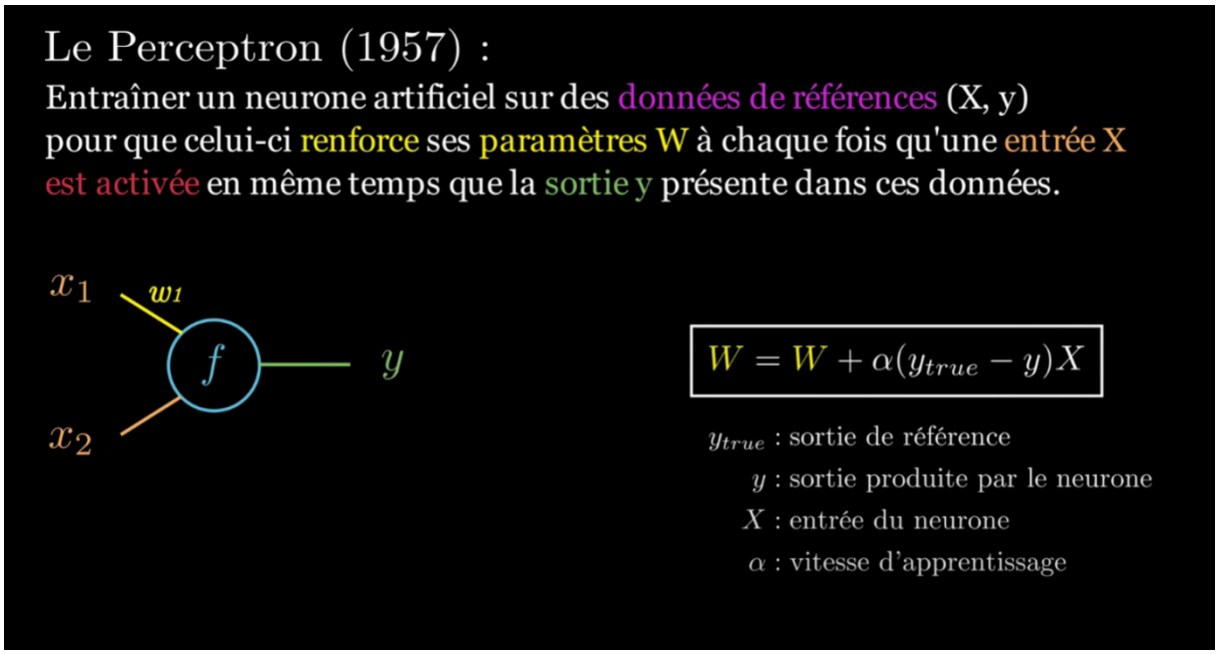
Cet algorithme est inspiré de la théorie de Hebb :

Lorsque 2 neurones biologiques sont excités conjointement, alors ils renforcent leur lien synaptique. En neurosciences, c'est ce qu'on appelle la plasticité synaptique et ça permet à notre cerveau de construire sa mémoire, d'apprendre de nouvelles choses ou encore faire de nouvelles associations.



Source : https://www.youtube.com/watch?v=XUFLq6dKQok&list=PLO_fdPEVlfKoanjvTJbIbd9V5d9Pzp8Rw&index=1

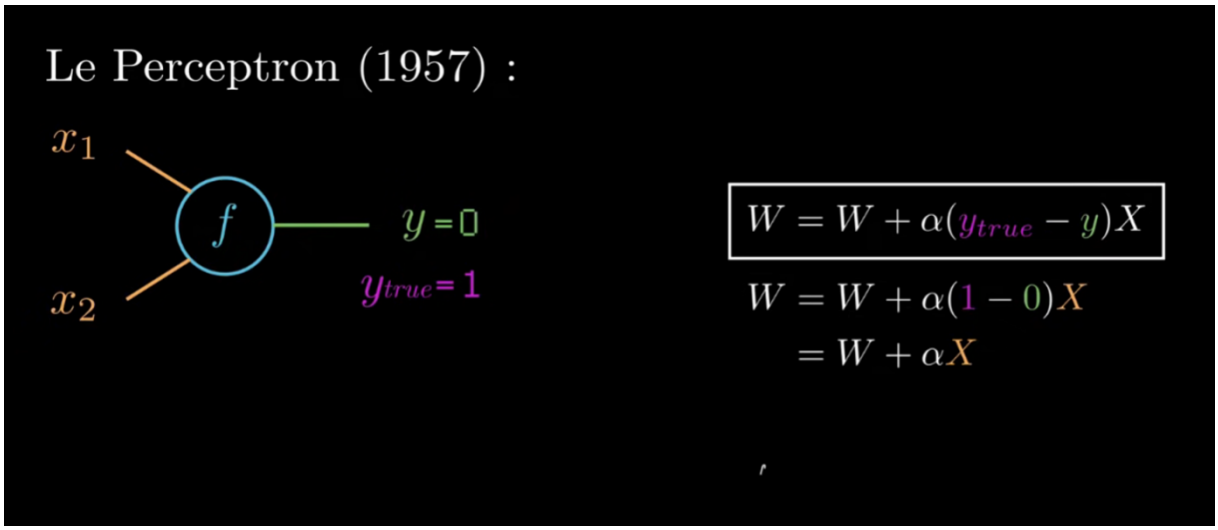
Frank a développé un algorithme d'apprentissage qui consiste à entrainer un neurone artificiel sur des données de références (X, y) pour que celui-ci renforce ses paramètres W à chaque fois qu'une entrée X est activée en même temps que la sortie y présente dans ces données.



Source : https://www.youtube.com/watch?v=XUFLq6dKQok&list=PLO_fdPEVlfKoanjvTJbIbd9V5d9Pzp8Rw&index=1

Pour ça, il a imaginé la formule suivante, dans laquelle les paramètres w sont mis à jour en calculant la différence entre la sortie de référence et à la sortie produite par le neurone, et en multipliant cette différence par la valeur de chaque entré X ainsi par un pas d'apprentissage positif.

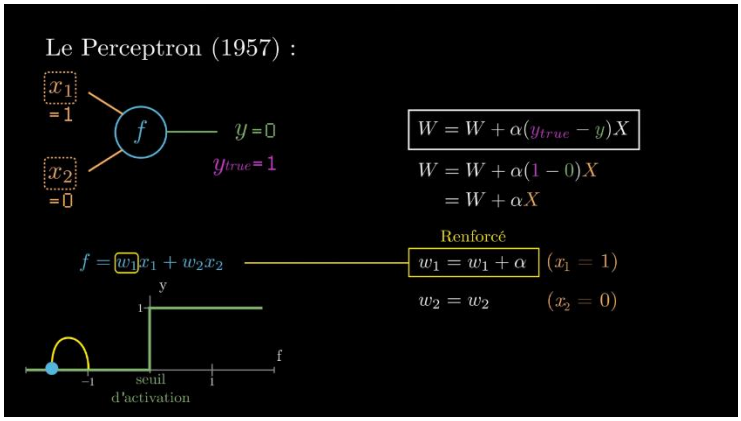
De cette manière, si notre neurone produit une sortie différente de celle qu'il est censé produire, par exemple s'il sort y = 0 au lieu de y = 1 alors notre formule nous donnera w = w + aX.



Source : https://www.youtube.com/watch?v=XUFLq6dKQok&list=PLO_fdPEVlfKoanjvTJbIbd9V5d9Pzp8Rw&index=1

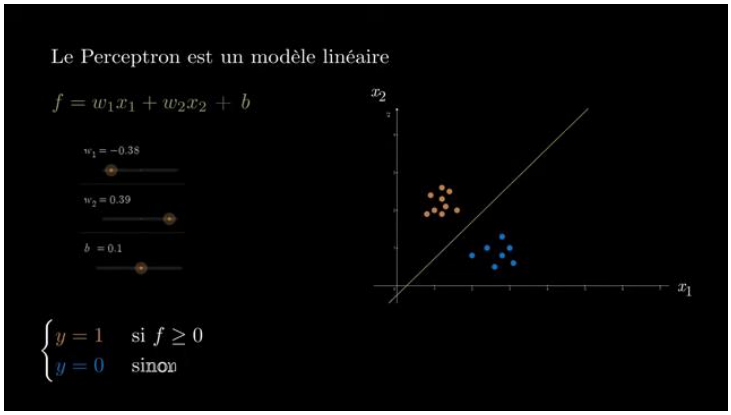
Donc, pour les entrées x qui valent 1, le coefficient w se verra augmenté d'un petit pas alpha, il sera « renforcé » (pour reprendre les termes de la théorie de Hebb) ce qui provoquera une

augmentation de la fonction w1 x1 + w2 x2 et qui rapprochera donc notre neurone de son seuil d'activation. Aussi longtemps que l'on sera en dessous de ce seuil, c'est-à-dire aussi longtemps que le neurone produira une mauvaise sortie, alors le coefficient w continuera d'augmenter grâce à notre formule, jusqu'au ou y_true vaudra y et à ce moment-là notre formule donnera w = w + 0. Ce qui fait que nos paramètres arrêteront d'évoluer.



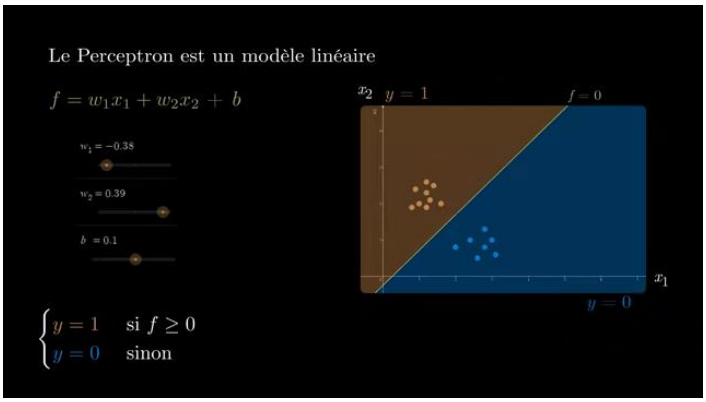
Source : *https://www.youtube.com/watch?v=XUFLq6dKQok&list=PLO_fdPEVlfKoanjvTJbIbd9V5d9Pzp8Rw&index=1*

Comme dit précédemment, le Perceptron est en fait un modèle linéaire, si l'on trace la représentation graphique de sa fonction d'agrégation **f (x1, x2) = w1x1 + w2x2** on obtient alors une droite, dont l'inclinaison dépend des paramètres w et dont la position peut être modifiée à l'aide d'un petit paramètre complémentaire qu'on appelle le biais.



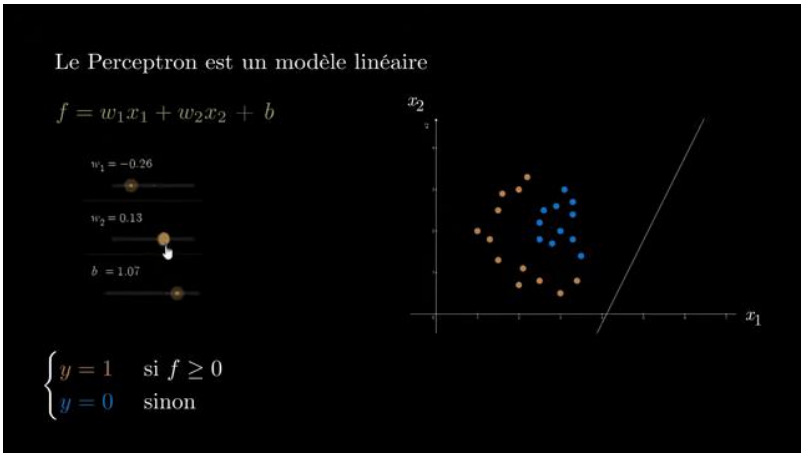
Source : *https://www.youtube.com/watch?v=XUFLq6dKQok&list=PLO_fdPEVlfKoanjvTJbIbd9V5d9Pzp8Rw&index=1*

Avec cette droite on peut par exemple séparer 2 classes de points, puisque grâce à notre fonction d'activation tout ce qui sera au-dessus de cette droite donnera une sortie y = 1 et tout ce qui sera en dessous donnera y = 0.



Source : https://www.youtube.com/watch?v=XUFLq6dKQok&list=PLO_fdPEVlfKoanjvTJbIbd9V5d9Pzp8Rw&index=1

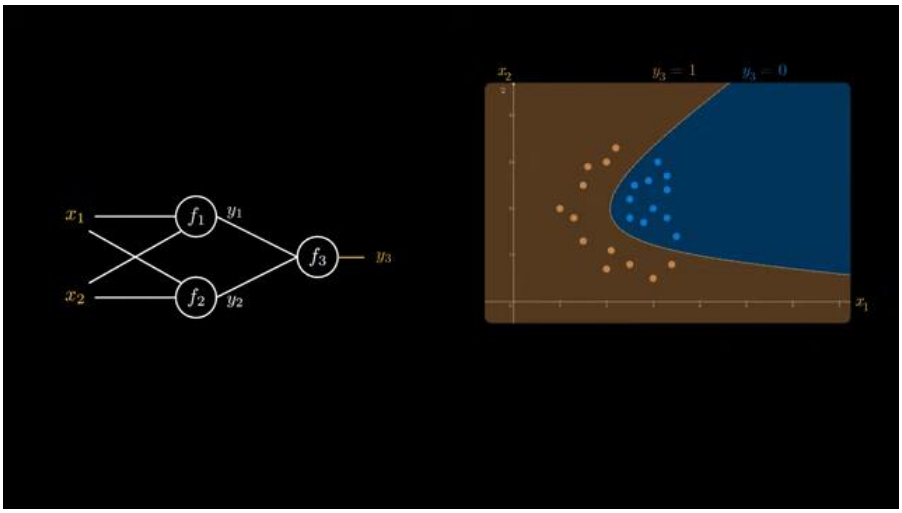
Le problème c'est qu'une grande partie des phénomènes de notre univers ne sont pas des phénomènes linéaires. Et des ces conditions, le permettront à lui seul n'est pas très utile.



Source : https://www.youtube.com/watch?v=XUFLq6dKQok&list=PLO_fdPEVlfKoanjvTJbIbd9V5d9Pzp8Rw&index=1

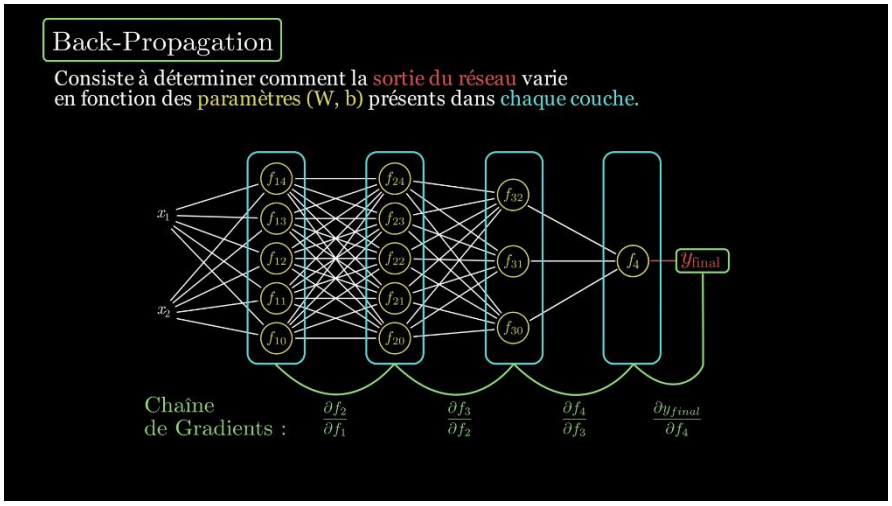
Mais McCulloch et Pitts avait réussi en connectant ensemble plusieurs neurones, à résoudre des problèmes plus complexes qu'avec un seul.

En rajoutant plusieurs couches au Perceptron on se demande comment entraîner un tel réseau de neurones pour qu'il fasse ce qu'on lui demande de faire, comment trouver les valeurs de tous les paramètres **w** et **b** de façon à obtenir un bon modèle ?



Source : https://www.youtube.com/watch?v=XUFLq6dKQok&list=PLO_fdPEVlfKoanjvTJbIbd9V5d9Pzp8Rw&index=1

La solution est d'utiliser une technique appelée Back Propagation, qui consiste à déterminer comment la sortie du réseau varie en fonction des paramètres présents dans chaque couche du modèle. Pour ça, on calcule une chaîne de gradients, indiquant comment la sortie varie en fonction de la dernière couche, puis comment l'avant dernière varie en fonction de l'avant dernière etc. ... jusqu'à arriver à la toute première couche de notre réseau. Il s'agit d'une Back Propagation : une propagation vers l'arrière.



Avec ces informations ont peut mettre à jour les paramètres de chaque couche, de telle sorte à ce qu'ils minimisent l'erreur entre la sortie du modèle et la réponse attendue. Pour ça on utilise on utilise une formule très poche de celle de Frank Rosenblatt, c'est la formule de la descente de Gradient.

Pour développer et entrainer des réseaux de neurones artificiels, ont répété en boucle les autre étapes suivantes :

- Foward Propagation

- Cost Function
- Back Propagation
- Gradient Descent

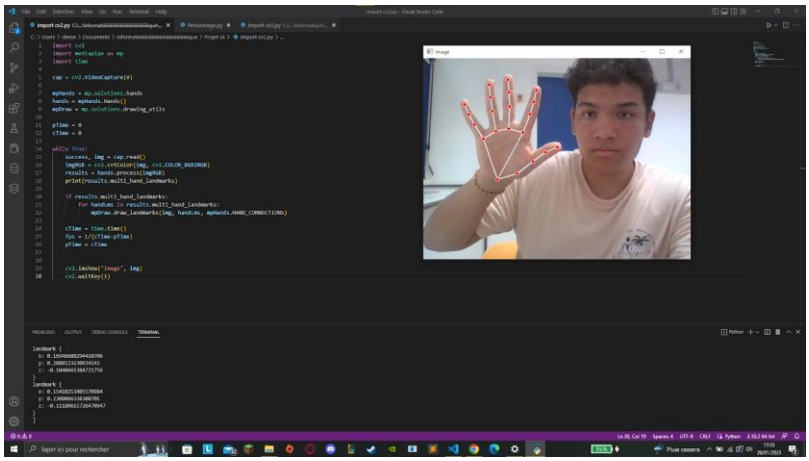
Maintenant qu'on a survoler la partie théorique sur le fonctionnement d'une intelligence artificielle, place a la création des IA !

IA avec la librairie mediapipe

Après plusieurs recherches nous sommes tombés sur la librairie Mediapipe que l'on à ajouter à un programme.

```
import cv2
import mediapipe as mp
import time

cap = cv2.VideoCapture(0)

mpHands = mp.solutions.hands
hands = mpHands.Hands()
mpDraw = mp.solutions.drawing_utils

pTime = 0
cTime = 0

while True:
    success, img = cap.read()
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    results = hands.process(imgRGB)
    print(results.multi_hand_landmarks)

    if results.multi_hand_landmarks:
        for handLms in results.multi_hand_landmarks:
            mpDraw.draw_landmarks(img, handLms, mpHands.HAND_CONNECTIONS)

    cTime = time.time()
    fps = 1/(cTime-pTime)
    pTime = cTime


    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

Mediapipe est une librairie qui détecte la position de la main comme ci-dessous :



La librairie permet de connaître le nombre de doigts levé en temps réel. Cependant en utilisant de la librairie de mediapipe on exploitait une base de données immense déjà créer ce pourquoi nous avons décidé de faire une autre IA avec notre propre base de données et notre propre réseau de neurone.

(Vous pouvez essayer l'IA dans dossier « IA avec la librairie mediapipe » et en lançant le script python « main.py »)

IA avec TensorFlow

A) Création de la base de données

Nous avons cherché à comprendre comment faire pour créer une base de données et l'entrainer afin qu'elle puisse donner les résultats escomptés. Nous avons donc utilisé cette vidéo :

Source : https://www.youtube.com/watch?v=yqkISICHH-U&t=5333s

Cette vidéo nous fournit les informations nécessaires pour créer notre propre base de données et comment l'entrainer avec des modèles déjà pré-fait.

Dans la vidéo il nous explique qu'il faut labeliser nos images avec un code python permettant de créer un fichier xml lier a l'image, le fichier xml contient le nom du label enregistrer ("un" par exemple) avec les cordonnées en y et x, et également la hauteur et la largeur.

Le programme est bien mais il faut labeliser les images une par une et c'est très long, on a donc décider de créer un programme qui labelise automatiquement et qui nous permet de gagner un temps fou. (Voir le programme autolabeling.py dans le dossier)

Le programme utilise la librairie mediapipie pour tracker la main et avoir la position de la main en x,y et en hauteur, largeur, avec toutes ces informations on peut créer manuellement un .xml et enchainer la création de la base de donnée.

Le programme divise la base de données en deux dossiers, "train" et "test" avec 80% de la base de données dans "train" et les 20% restant dans "test".

B) Entrainement et évaluation de l'IA

Dans un deuxième temps nous allons train l'IA à partir d'un modèle de réseau de neurone déjà créer. Il va utiliser le dossier train pour s'entrainer. Un checkpoint va être créer, par la suite il faut évaluer l'IA, maintenant il va utiliser le dossier test pour s'évaluer.

C) Conclusion de l'IA

L'IA fonctionne correctement mais sous certaines conditions, le plus souvent avec le même fond que les photos prise pour l'apprentissage de l'IA. De plus l'IA n'as pas était entrainer avec notre propre réseau de neurone mais un réseau de neurone pré-fait. On c'est donc tourner vers une autre IA

(Vous pouvez essayer l'IA dans dossier « IA avec TensorFlow » et en lançant le script python « main.py »)

IA avec TensorFlow et mediapipe

A) Le principe de l'IA

Pour commencer nous avons eu l'idée de combiné la librairie mediapipe et TensorFlow dans la même IA. Ici au lieu de prendre des photos de nos mains vierges on va ici intégrer mediapipe qui va tracer des traits sur les mains et on va prendre les photos avec les traits sur les mains, avec ça l'IA aura moins de difficulté à reconnaitre la main.

Nous avons également créé notre propre réseau de neurone.

B) Entrainement et évaluation de l'IA

Pour l'entrainement et l'évaluation de l'IA ça se passe de la même manière que l'ancienne IA, la seule différence et que l'entrainement s'effectue avec notre propre réseau de neurone.

C) Conclusion

L'IA avec Tensorflow et mediapipe est l'IA qui fonctionne le mieux et ça de loin, l'apparition des traits de la main aide énormément l'IA à reconnaitre la main, avec une réussite de 98%.

(Vous pouvez essayer l'IA dans dossier « IA avec TensorFlow + Mediapipe » et en lançant le script python « main.py »)