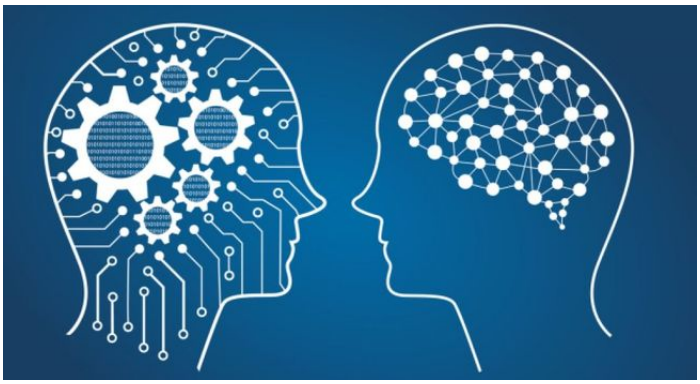
Université de Montpellier

Rapport de TER L2

Apprentissage automatique et applications



Louis Lamalle , Boyan Bechev , Romain Jaminet , Hugo Gianfaldoni , Romain Fournier

Tuteur : M.SERIAI

1er Janvier 2019 — 30 Avril 2019

Table des matières

1 Présentation du sujet

1.1 Introduction

Dans le cadre du projet de programation de L2 , nous avons décidé de nous pencher sur le domaine de l'intelligence artificielle , et plus particulièrement celui du machine learning. Nous redétaillerons plus tard ces termes dans ce document. Notre sujet étant axé sur la recherche , nous avons eu une grande liberté pour les choix des domaines d'explorations. Pour que nous puissions avoir un champ d'exploration le plus large possible, nous avons travaillé sur trois différentes applications du machine learning :

- La reconnaissance d'image.
- La reconnaissance de son.
- Simulation de voiture autonome.

Ces trois axes de recherche nous ont été suggérés par notre encadrant pour que nous puissions acquérir un champ de vision aussi large que possible.

Ce projet a été réalisé par Louis LAMALLE, Boyan Bechev, Romain Jaminet, Hugo Gianfaldoni et Romain Fournier. Monsieur Abdelhak-Djamel SERIAI à été notre encadrant durant ce projet.

1.2 Motivation

Nous voulions etudier ce sujet car le domaine du machine learning nous intéresse fortement. En effet , le machine learning , et plus particulièrement le deep learning , est une branche en pleine expansion depuis quelques années. Qui n'a pas entendu parler de la victoire d'AlphaGo , l'intelligence artificielle de Deepmind , sur les plus grands champions de go ? Qui ignore encore la révoution des voitures autonomes de Tesla ou Uber ? C'est d'abord cette mise en avant médiatique qui à piqué notre curiosité.

De plus l'intelligence artificielle fait l'objet de nombreux fantasmes comme de nombreuses craintes bien retranscrites dans les films de sciences fictions , comme Terminator par exemple.

Dans un second temps , ce sujet à suscité notre interêt car il est axé sur la recherche dans un domaine que nous ne connaissons que très peu dans le cadre de la fac puisqu'il n'est pas traité.

C'est principalement pour ces deux raisons que nous avons voulu travailler sur ce projet , dont nous vous exposons le rapport ci dessous.

1.3 Organisation générale

Etant donné nos faibles connaissances en la matière, nous avons débuté notre travail par deux semaines de recherches sur l'intelligence artificielle et le machine learning grâce, notamment, aux documents fournis par notre encadrant.

Dans un second temps , étant donné la durée relativement courte pour réaliser le projet , nous nous sommes divisés en trois groupes. Le premier était chargé d'étudier la reconnaissance d'image , le second sur la reconnaissance de son , et le dernier sur une simulation d'une voiture autonome.

Pour partager et communiquer sur nos avancées respectives , nous avons utilisés l'application Discord. De plus le groupe se réunissait toutes les semaines en plus de la réunion hebdomadaire avec l'encadrant.

Pour organiser notre travail dans le temps nous avons utilisé un calendrier Google pour fixer les réunions et autres échéances.

Pour tout ce qui concerne le partage des documents , nous avons utilisés un dépot git , ce qui nous à été très utile dans le cadre d'un projet colaboratif.

2 Etat de l'art

2.1 Qu'est ce que l'intelligence artificielle ?

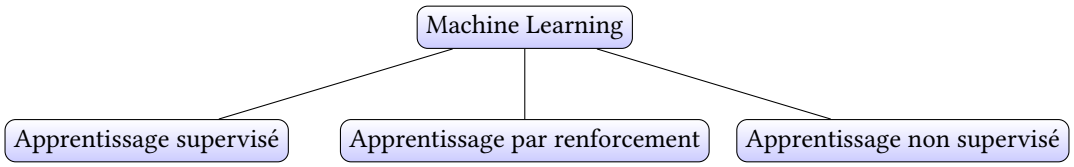
L'intelligence artificielle (IA, ou AI en anglais pour Artificial Intelligence) consiste à mettre en œuvre un certain nombre de techniques visant à permettre aux machines d'imiter une forme d'intelligence réelle. L'IA se retrouve implémentée dans un nombre grandissant de domaines d'application.

La notion voit le jour dans les années 1950 grâce au mathématicien Alan Turing. Dans son livre Computing Machinery and Intelligence, ce dernier soulève la question d'apporter aux machines une forme d'intelligence. Il décrit alors un test aujourd'hui connu sous le nom « Test de Turing » dans lequel un sujet interagit à l'aveugle avec un autre humain, puis avec une machine programmée pour formuler des réponses sensées. Si le sujet n'est pas capable de faire la différence, alors la machine a réussi le test et, selon l'auteur, peut véritablement être considérée comme « intelligente ».

[1]IA

2.2 Qu'est ce que le machine Learning

L'apprentissage automatique (en anglais machine learning, littéralement « l'apprentissage machine ») ou apprentissage statistique est un champ d'étude de l'intelligence artificielle qui se fonde sur des approches statistiques pour donner aux ordinateurs la capacité d' « apprendre » à partir de données, c'est-à-dire d'améliorer leurs performances à résoudre des tâches sans être explicitement programmés pour chacune. Plus largement, cela concerne la conception, l'analyse, le développement et l'implémentation de telles méthodes. On peut séparer le machine Learning en 3 branches distinctes :



On ne s'intéresse ici qu'a l'apprentissage supervisé et l'apprentissage par renforcement (ceux utiles aux programmes).

[2]Wiki

2.3 L'apprentissage supervisé

L'apprentissage supervisé consiste à entrainer une machine dans le but de lui faire apprendre une fonction de prediction pour la résolution d'un problème.

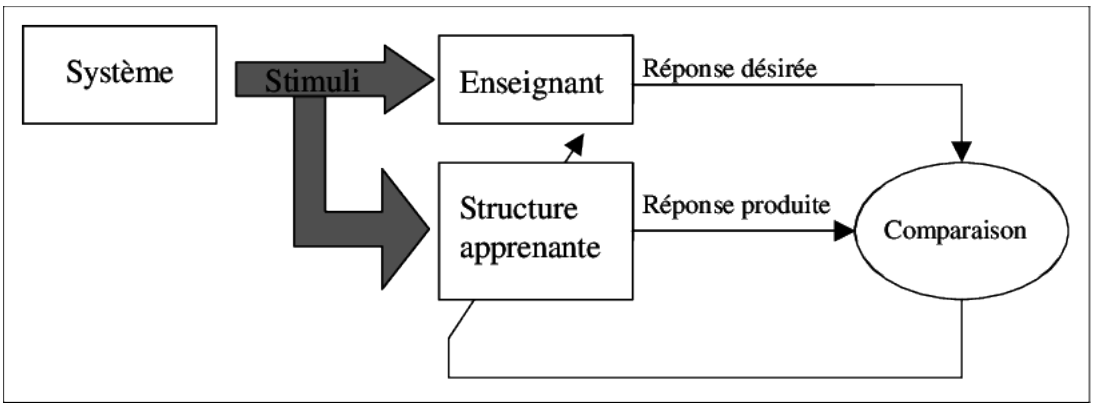
Pour se faire , on fournit à la machine une base de donné étiqueté. L'étiquette indique à l'algorithme ce qu'il doit avoir en sortie en fonction de la donnée qui lui à été donné en entré. Grâce à différents algorithmes , le modèle peut "apprendre de ses erreurs" pour ensuite avoir un bon taux de prédiction.

2.4 Classification

La classification au sens large correspond à l'organisation d'entité en différentes catégories sel.on certains critères définis

En machine learning , on se sert de la classification pour prédire une donnée qualitative d'un objet.

On entraine la machine avec une base de données étiqueté pour classer les données en différentes "classes". On pourra donc prédire la "future étiquette" d'un objet en fonction des attibuts de l'objet.



La "structure apprenante" décrite sur le schéma est souvent un réseau neuronal (voir plus bas).

[3]Lien image

2.5 Apprentissage par renforcement

Le reinforcement learning, abrévié "RL", est une branche du domaine du machine learning. Il met en oeuvre un modèle représenté par un "agent" et prédit des sorties en fonction d'entrées . le RL est utilisé lorsqu'un agent doit évoluer dans un environnement, son apprentissage est dicté par une fonction qui par un système de point peut lui attribuer une récompense lorsqu'il réalise une bonne performance ou une bonne action. (fig. 1)

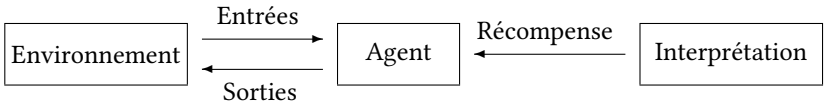


Figure 1 – Principe de fonctionnement

Un agent est constitué d'une partie qui traitera les entrées que l'environnement lui fournis et engendrera des signaux en sortie, et d'un mécanisme d'apprentissage dicté par la méthode choisie. Il est possible de représenter la partie traitante de l'agent avec un tableau d'état et d'actions possibles (comme avec la méthode "Q-learning") mais ce rapport se penchera sur les approches à réseau de neurones.

2.6 Réseau de neurones

Un réseau de neurone, appelé "Neural network" en anglais est une structure qui s'appuie sur le fonctionnement biologique de neurones. Un réseau de neurones permet de traiter une information de manière probabiliste, et de générer des sorties en fonction des entrées. un réseau neuronal couplé avec les différentes méthodes d'apprentissage, permet à l'agent d'apprendre à résoudre un problème donné. Le plus souvent, le réseau est traité comme une boite de pandore et les seules interactions directes avec l'environnement sont les entrées et les sorties. Grâce à différentes modifications du réseau au cours de l'apprentissage , ce dernier constituera un modèle permettant d'effectuer les tâches pour lesquelles il est entrainé. (fig. 2)

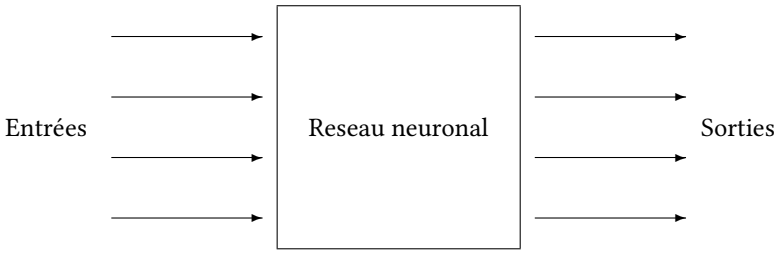


Figure 2 – Reseau de neurone

2.6.1 Le neurone

Un neurone est composé de une ou plusieurs entrées et d'une sortie qui correspond a la somme de ses entrées passées par une fonction. Il en existe plusieurs types comme le neurone "simple" ou suivant une table de vérité, mais le plus communément utilisé est le neurone suivant le modèle McCulloch et Pitts dit "MCP" (fig. 3). Ce neurone met à l'échelle ses entrées $E_1, E_2, ..., E_n$ grâce aux valeurs dites "poids" $P_1, P_2, ..., P_n$, en fait la somme et les compare a une valeur dite "seuil" T. Mathématiquement, un neurone MCP émet un signal si la formule suivante est vérifiée.

$$E_1P_1 + E_2P_2 + ... + E_nP_n > \mathrm{T}$$

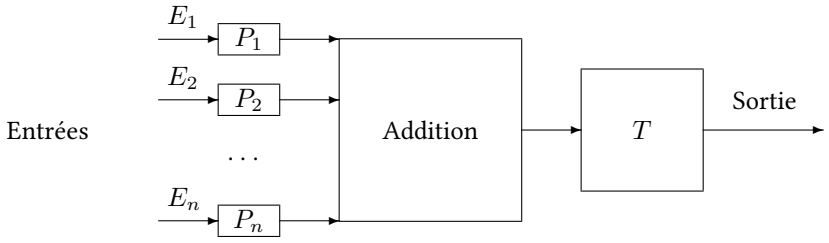


Figure 3 – Neurone "MCP"

En modifiant les paramètres P et T des neurones, le comportement du réseau peut être altéré. C'est par ce procédé que l'agent peut apprendre et évoluer.

2.6.2 Le réseau

Un réseau de neurone est une multitude de neurones connectés les uns aux autres. Un réseau possède une couche de neurones d'entrées, une couche masquée représentant la plus grande partie du réseau, et une couche de sortie. Il existe deux grand types de réseau, les reseaux "feedback" acceptant que les neurones puissent former des boucles, et les réseaux "feed-forward" qui arrange les neurones en couches et qui permet aux neurones d'être connectés seulement à une couche supérieure (fig. 4). On notera que la sortie d'un neurone peut être connecté a plusieurs neurones, il s'agit d'une sortie dupliquée et non pas de plusieurs sorties indépendantes.

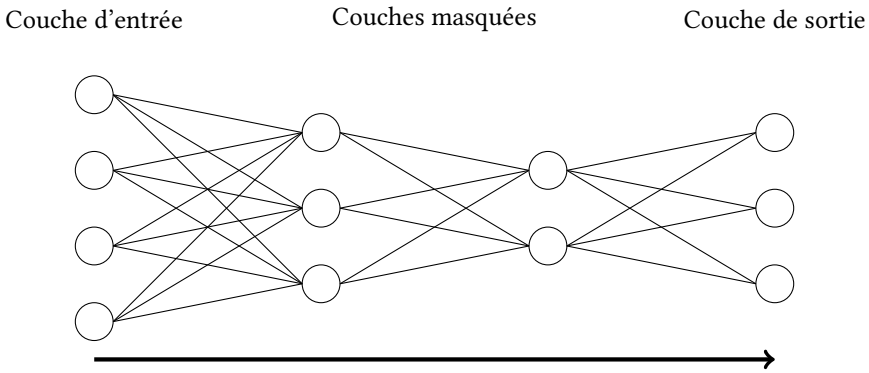


Figure 4 – Réseau de neurones "feed-forward"

3 Les projets

3.1 La reconnaissance d'expressions faciales

3.1.1 La reconnaissance faciale dans le monde

La reconnaissance faciale connaît un succès fulgurant grâce à la démocratisation de l'intelligence artificielle et à un accès simplifié à beaucoup de puissance de calcul.
Tout d'abord, les smartphones les plus récents possèdent presque tous un module de déverrouillage à l'aide de la reconnaissance faciale. Facebook et Google ont développé des algorithmes ayant une meilleur précision qu'un être humain pour reconnaître sur deux images différentes un même individu. L'état américain utilise la reconnaissance faciale à des fins de sécurité et plus, notamment dans les aéroports.
La reconnaissance faciale est également utilisé pour détecter des maladies génétiques rares et faire le suivi psychologique de patient potentiellement fragile (dépression, schizophrénie...).
Il ne s'agit là que de quelques exemples de ce que la reconnaissance faciale peut effectuer grâce à l'intelligence artificielle.

3.1.2 Les solutions possibles [À compléter]

Ce que nous devons réaliser, c'est la modification et l'entraînement d'une intelligence artificielle déjà existante pour quelle puisse reconnaître les expressions faciales d'un être humain sur une image.
Pour cela, différents algorithmes et systèmes s'offrent à nous :

1. La méthode Eigenface
2. La méthode Fisherface
3. L'analyse discriminante linéaire
4. Le modèle de Markov caché
5. Quelques réseaux de neurones de reconnaisance d'objet en temps réel
   (a) Retinanet (utilisant tensorflow)
   (b) YoloV3 (utilisant darknet)

3.1.3 Explication de la solution choisie [À compléter]

Nous avons choisi d'utiliser YoloV3 comme base pour l'apprentissage de notre système de reconnaissance d'expression faciale car il affichait des performances très intéressantes et donc pouvais exploiter la caméra en temps réel.
Fonctionnement de YoloV3 : (à compléter)
https ://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b
https ://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b
Yolo fonctionne de la manière suivante : [à compléter]

1. On configure le réseau de neurone que nous allons utiliser (poids initiaux, couches, filtres...)
2. On donne la liste des objets que nous voulons qu'il détecte
3. Pour chaque image il faut créer un fichier texte contenant les informations suivantes :

   — <Numéro du label> <centre x> <centre y> <largeur> <hauteur>

4. On démarre l'entrainement

Pour utiliser YoloV3 à des fins de reconnaissance d'expressions faciales, il faut créer une classe par expression faciale, par exemple :

1. Neutre
2. Content
3. Triste
4. En colère
5. Peureux
6. Surpris
7. Autre

3.1.4 Configurations et entraînements [À compléter]

1. Plan d'action
   Pour entrainer Yolo à reconnaître des expressions faciales, j'ai utilisé un fichier de configuration qui est basé sur YoloV3 pour la reconnaissance d'objet.
   J'ai réussi à obtenir une base de données d'image de 122Go soit environ un million d'image provenant en contactant l'université de Denver.
   Il y a 11 labels différents : Neutre,Content,Triste,Surpris,Peur,Dégoût,Colère,Mépris,Rien,Incertain,Pas de visage, cependant il y a un problème, il faut que pour chaque image, je définisse le centre du rectangle du visage, ainsi que la largeur du rectangle et sa hauteur ce qui est vraiment dommage pour une base de données de cette taille.
   Je vais donc devoir me limiter, je vais commencer par ne garder que quelques labels comme par exemple : Neutre,Content,Colère,Triste.
   Je ne vais travailler que sur une cinquantaine d'image par label pour commencer soit environ 200 images. Je ne suis pas sûr que cela suffira pour la précision et la sureté, il s'agit d'un essai.
   Il faut également que je convertisse chaque image en .jpg et que je les redimensionne à la même taille. Vu que les images sont carrées, il n'y aura pas de déformation visuelle, juste éventuellement une perte de pixel qui n'est pas importante, car les images sont similaires en taille.

   Les labels sont dans un fichier csv, je vais donc faire un script en python me permettant de les récupérer, comme je n'ai pas pu télécharger toutes les images, je vais simplement faire un script qui va agir de la sorte :
   Pour chaque image dans le dossier des images :
   (a) Si l'image labélisé correspond à : content,triste,neutre,colère alors
      i. On créer un fichier texte nommé par le nom de l'image en .txt contenant le numéro du label
      ii. J'utilise face detector (qui utilise tensorflow) pour détecter la position du visage et je vais rajouter à mon fichier les informations manquantes

   La labélisation est assez lente donc je ne peux pas traiter beaucoup d'image à moins de laisser tourner mon ordinateur longtemps, je vais utiliser un VPS pour faire la labélisation à distance sans interruption.

   face_detector : https ://github.com/ageitgey/face_recognition

2. Le premier entraînement
   (a) Architecture du réseau de neurone
   (b) Entraînement
      Après avoir labélisé 500 images à l'aide d'un script en python que j'ai développé, utilisant la librairie "face detector" (il s'agit d'une IA qui permet de tracer les rectangles autour des visages), j'ai par la suite déporté darknet sur Windows pour pouvoir l'utiliser avec ma carte graphique Nvidia, pour profiter des technologies CUDA et cuDNN. Le premier entraînement c'est donc effectué sur 500 images, ce qui est relativement peu, les développeurs de darknet conseillent environ 2000 données par classe donc il me faudrait environ 8000 images pour le moment avec les 4 classes (neutre, content, triste, colère) et 22000 si je veux utiliser tous les labels de la base de données (qui contient 1 million d'image).
      Après avoir entrainé le réseau de neurones sur 500 images pendant environ 30 minutes, le taux de perte était inférieur à 1 donc c'était correct. J'ai donc fait des tests avec ce nouveau réseau de neurone entrainé, avec de nouvelles images et les résultats sont très correcte sur des images qui sont calibré dans les mêmes dimensions que les images d'entrainement, cependant le taux de certitude n'est pas assez élevé (environ 50%).
      Cela est surement lié au temps d'apprentissage qui n'était pas très élevé et au nombre de données.

   (c) Résultat
3. Le deuxième entraînement
   (a) Architecture du réseau de neurone
   (b) Entraînement
      Le deuxième entrainement a été effectué avec un jeu de données de 2500 images, ce n'est toujours pas assez mais cela est tout de même 5 fois supérieur au premier jeu de données. Le taux de certitudes est désormais plus élevé (environ 80%).
      Pour avoir un taux proche de la perfection et une reconnaissance du visage instantanée, il faudrait que je diversifie mes données d'entrainement et que j'en augmente encore le nombre.
   (c) Résultat
4. Le troisième entraînement

(a) Architecture du réseau de neurone

(b) Entraînement
Pour le troisième entraînement, j'ai ajouté une classe, la classe "Pas de visage", car l'intelligence pouvait se laisser avoir par des photomontages où par exemple un fruit prenait l'apparence d'un humain avec une bouche et des yeux. J'ai également laissé executé la labélisation sur un VPS durant environ 20 heures, ce qui m'a permit de labéliser exactement 30664 images. Cependant la répartission n'est pas égale :
— Neutre : 8123
— Content : 11887
— Triste : 1270
— Colere : 2231
— Pas de visage : 7153

(c) Résultat

5. Le quatrième entraînement

(a) Architecture du réseau de neurone

(b) Entraînement

(c) Résultat

darknet pour windows : https ://github.com/AlexeyAB/darknet

A modifier et compléter

3.1.5 Résultats et perspectives [À faire]

A compléter

3.2 La reconnaissance vocale

3.2.1 Le problème de la reconnaissance automatique de la parole

La reconnaissance automatique de la parole (que l'on appellera vulgairement aussi "reconnaissance vocale") est un axe majeur de recherche dans le domaine du machine learning. Ce problème est inclus dans le domaine du traitement de la parole qui concerne , rappelons le , tout les problèmes liés à la captation et la transmission de la parole (comme par exemple les transmissions téléphoniques).

La reconnaissance automatique de la parole est un enjeu majeur pour les interfaces hommes-machines au vu de la multiplication des objets connectés et autres technologies utilisant la reconnaissance vocale. Ces technologies sont aussi bien présentes dans nos maison (assistant personnel intelligent comme Google Home ou Alexa de Amazon) que dans les usines (l'assistant vocal Athéna , développé par ITSPEEX est l'équivalent d'Alexa pour les usines) . Elle est également présente auprès des plus jeunes avec la commercialisation de robots jouets équipés de reocnnaissance vocale. Cette omnisprésence prouve que la reconnaissance automatique de la parole est un enjeu scientifique et surtout économique primordial.

Pour ces raisons , nous avons décidés de nous intéresser à cette application du machine learning. Nous allons donc essayer de concevoir un modèle , grâce à un algorithme de machine learning , permettant la classification d'un ensemble de quelques mots. Nous utiliserons un réseau de neurones dans le cadre d'un apprentissage supervisé.

Pour rappel, la classification est le fait d'attribuer chaque objet (ici un enregistrements vocal qui contient une voix quelconque énonçant un mot parmis ceux que l'on souhaite classer) à une classe (une classe correspond à un mot parmi ceux à classer).

Nous allons tout d'abord vous exposer les outils que nous avons utilisés pour mener à bien notre projet. Nous allons par la suite détailler le travail de recherche que nous avons effectués . Avant de conclure , nous présenterons les résultats obtenues et les possibles améliorations.

[4]Athena [5]Google Home [6]Wiki Reconnaissance vocale

3.2.2 Outils et langages

De nombreux outils sont disponibles pour implémenter des algorithmes de machines learning. Cependant , ce domaine n'étant pas le plus commun et son développement plutôt récent , ces outils sont assez peu utilisés et documentés sur internet ce qui rend le travail de recherche fastidieux. Nous présentons ci dessous les langages et outils de calcul que nous avons utilisés.

Langage de programmation : python

Pour ce projet , nous avons décider de programmer avec le langage de programmation python qui est un langage de programmation interprété de haut niveau. Nous l'avons choisi pour différentes raison :

- Langage de haut niveau : Python est un langage de plus haut niveau que le C (mais plus lent). Il est donc plus facile d'utilisation et nous permet de nous focaliser plus sur la manipulation des données et de notre modèle plutôt qu'à la gestion de mémoire.
- Manipulation des fichiers : La manipulation des fichiers en python (ouverture, parcours de dossier ...) est très simple.
- Les librairies : De nombreuses librairies sont disponibles pour créer des algorithmes de machine learning comme par exemple Pandas , Agate ou Tensorflow. Il existe aussi des bibliothèques permettant de manipuler les tenseurs (utilisés en machine learning et surtout par TensorFlow) comme NumPy.

[7]python [8]tenseur

Librairies : Tensorflow et TFlearn

Pour implémenter des réseaux de neurones dans le cadre du machine learning , nous avons deux possibilités :

- Implémenter l'algorithme "from scratch" (cf projet de simulation de voiture autonome)
- Utiliser des librairies permettant d'implémenter plus facilement les différents type de réseaux de neurones.

Pour ce projet nous avons pris la décision d'utiliser la librairie TFlearn qui utilise le framework Tensorflow.

TensorFlow ™ est une bibliothèque de logiciels open source pour le calcul numérique haute performance. Son architecture flexible facilite le déploiement de diverses plates-formes. Développé à l'origine par des chercheurs et des ingénieurs de l'équipe Brain au sein de l'entreprise Google, il intègre un support puissant pour l'apprentissage automatique et

l'apprentissage en profondeur. Le cœur du calcul numérique flexible est utilisé dans de nombreux autres domaines scientifiques. (Pour plus d'informations sur TensorFlow , cf rapport annexe)

TFlearn est une librairie d'apprentissage profond basée sur TensorFlow. Elle permet de faciliter l'implémentation de différent modèles de deep learning comme les réseaux de neurones récurrent ou les réseaux neuronaux convolutifs. C'est pour cette simplicité que nous utilisons cette librairie. Elle permet notamment de créer des réseaux de neurones couche par couche de manière très rapide :

```
net = tflearn.input_data([None, width, height])
net = tflearn.lstm(net, 128, dropout=0.8)
net = tflearn.fully_connected(net, classes, activation='softmax')
net = tflearn.regression(net, optimizer='adam', learning_rate=learning_rate, )
```

[9]TFlearn [10]TensorFlow

Outils de calculs

Le machine learning et plus particulièrement le deep learning, demande d'importantes ressources de calculs. Faire fonctionner les algorithmes d'apprentissage sur un laptop présente plusieurs inconvénients comme la lenteur des calculs et l'usure du matériel. Pour cela plusieurs solutions sont possibles pour entrianer les modèles de machines learning :

- Serveur Cloud : Il existe des serveurs spécialement dédiés au machine learning utilisant des GPU , plus adaptés au calcul matriciel utilisé pour l'apprentissage profond (Google TPU, AWS d'Amazon ...)
- Utilisation de la carte graphique : Des solutions sont possibles pour utiliser la carte graphique de son laptop pour effectuer les calculs (technologie CUDA avec les GPU NVIDIA)

[11]CUDA [12]Google TPU

3.2.3 Travaux de recherche

Pour comprendre comment implémenter un algorithme de machine learning pour la reconnaissance automatique de la parole, nous nous sommes donnée l'objectif de classer 10 mots différents (voir ci dessous). Il est important de préciser que plusieurs "versions" de ce projet ont été réalisé utilisant différents modèles d'apprentissage et différentes représentation des données. Ici il ne sera présenté que la version ayant aboutie aux résultats les plus intéressants. Il sera toutefois mentionné les différences les plus importantes des autres versions. Nous utiliserons un algorithme de deep learning (voir plus haut) combiné à un dataset conséquent contenant des clips vocaux.

Représentation des données et des labels

Pour obtenir un système de reconnaissance vocale efficace , il est nécessaire d'avoir un dataset de qualité :

- Pour chaque classe de mot , il doit y avoir un nombre important de fichiers qui serviront à entrainer notre modèle.
- Les données doivent être suffisament diversifié pour eviter le problème de "surapprentissage". En effet si les données sont trop similaires, le modèle reconnaitrai par coeur les données sur lesquels il est entrainé mais serait incapable de "predire" la classe d'un mot.
- Les données doivent être correctement labélisées. Comme expliqué précédemment, l'apprentissage supervisé à besoin de données, passées en entré du réseau neuronal, et un label "objectif" qui permettra de calculer l'erreur entre la sortie du réseau et cet objectif pour ensuite mofifier le réseau. Ici , les labels sont crées en prenant le premier chiffre de chaque fichier qui correspondra à sa classe et en les mettant dans un tableau. Par exemple tout les fichiers audio correspondant au mot "house" commencera par le chiffre 1.

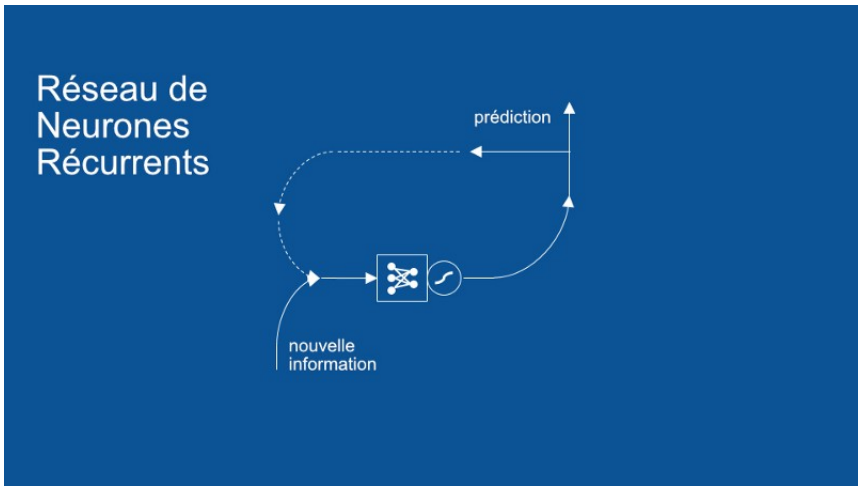
Nous cherchons à classifier les mots suivants : "dog" , "house" , "on" , "stop" , "sheila" , "marvin" , "wow" , "left" , "cat" , "go" . Nous téléchargeons donc un ensemble 20000 clips vocaux (soit 2000 enregistrements par mots).

Une fois les données téléchargées , il faut convertir les fichiers wav (WaveForm Audio File Format) en structure de données exploitables par notre modèle pour l'entraîner. La librairie librosa permet de représenter les fichiers audio en coefficient MFC qui représente la densité spectrale de puissance. Ces coefficients sont placés dans des tableaux qui pourront être exploités par la suite.

NB : Les mots ont été choisi en fonction de la disponibilités des données sur internet.

Le réseau de neurones

Avant d'expliquer le fonctionnement de l'entrainement , voici le detail du réseau de neurones utilisé. C'est un réseau de neurones récurrent , c'est à dire que le réseau va "enregistrer" la prédiction précédente pour pouvoir optimiser la prédiction courante (cf l'image ci dessous). Nous utilisons un réseau de neurones récurrent car il est le plus efficace pour les problèmes de reconnaissance de la parole (Nous avons également créé un modèle avec un réseau de neurones convolutif avec des résultats intéressants également).



Nous utilisons un réseau à 4 couches :

- Une couche d'entrée : cette couche sert excusivement de "porte d'entrée" pour les données.

— Une couche long short term memory (lstm) : c'est la couche récurrente du réseau. La particularité des réseaux lstm est leur faculté à pouvoir enregistrer plusieurs des prédictions précédentes utiles et non seulement une comme le fait un réseau neuronal de base.
— La troisième couche est une couche complètement connecté. Tout les neurones de cette couches sont donc connecté à tout les neurones de la couche précédentes.
— La dernière couche est une couche de regression. Elle va effectuer une régression (expliqué précédemment) pour prédire le résultat final (ici le mot correspondant). On inclut l'optimiseur "Adam" dans cette couche. Le rôle de l'optimiseur est de modifier les coefficients du réseau de neurones en fonction de l'erreur obtenue. C'est ce qui va nous permettre de réduire le nombre d'erreur en fonctions des entrainements.

Exemple de l'implémentation de la couche lstm en python :

```
net = tflearn.lstm(net, 128, dropout=0.8)
```

[13]Plus sur les RNN [14]Lien de l'image

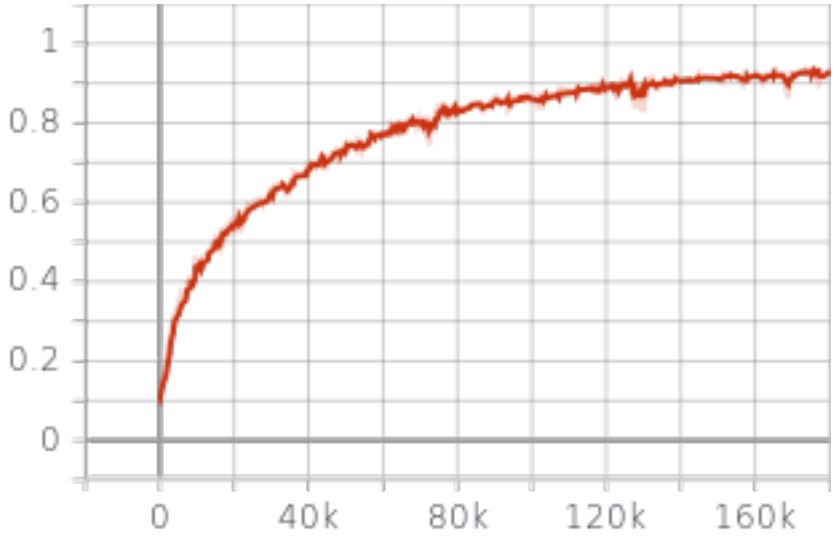
L'entrainement

Pour entrainer le modèle, nous utilisons la fonction fit de la librairie TFlearn :

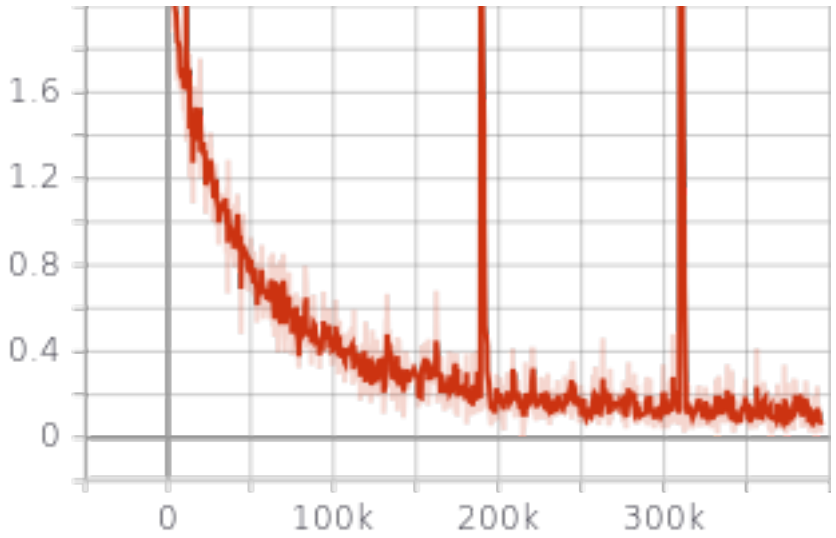
```
model.fit(trainX, trainY, n_epoch=1000, show_metric=True, batch_size=64,
                                                    snapshot_step=500)
```

Le processus d'entrainement est l'étape la plus longue. Elle consiste à modifier le réseau de neurones jusqu'à ce qu'il aie un taux de prédiction correct suffisamment élevé. Voici les différentes étapes :
— Un ensemble de 64 fichiers (appelé batch) va être envoyé au réseau de neurones. Ce dernier va essayer de prédire les mots qui lui sont donnés.
— Une erreur va ensuite être calculée par rapports aux prédictiosn si elles sont bonnes ou correct.
— Cette erreur va être rétropropagée pour modifier les coefficients du réseau de neurones. C'est le rôle de l'optimiseur Adam.
— On réitère l'opération jusqu'a ce que tout les fichiers soient passés au modèle. (Cela correspond à une époque)
— On entraine le réseau sur 500 époques.

Au fur et à mesure de l'entrainement , le "loss" (indicateur de mauvaise prédiction) diminue et la précision augmente. Voici ce les graphes produits par TensorFlow au travers de son outil de virtualisation TensrBoard lors de l'entrainement :



Représentation de la précision en fonction du nombre d'étapes d'apprentissage.



Représentation du loss en fonction du nombre d'étapes d'apprentissage.

3.2.4 Résultats obtenus et conclusion :

Pour tester les performances de notre modèle , nous devons lui présenter des fichiers audio sur lesquels il ne s'est pas entrainé et regarder si les prédictions sont exactes. On observe que les résultats obtenus sur des enregistrements réalisés par nous-même sur ordinateur sont moyens. Cependant les résultats obtenus avec des fichiers audios téléchargés sur internet sont très bons. Les seules erreurs sont sur les mots "on" et "go" , sûrement à cause de leur proximité. On pense donc que les mauvais résultats obtenus sur nos propres fichiers audios sont dûs à la mauvaise qualité d'enregistrement. Pour résoudre ce problème , il est possible de rajouter du "bruit" sur les fichiers d'entrainements. Pour augmenter la précision du modèle il est également possible d'augmenter le nombre de fichiers d'entrainements.

3.3 L'apprentissage de la conduite autonome

Ce travail à pour but d'explorer la partie "Reinforcement Learning" du projet. Le concept d'apprentissage par renforcement étant relativement intuitif, et les projets étant suffisamment abondants sur internet, nous avons choisi de coder notre propre implémentation d'un algorithme de Reinforcement Learning en y voyant une opportunité de mieux comprendre le fonctionnement de cette méthode d'apprentissage.

L'apprentissage par renforcement posant des barrières comme l'absence de données d'apprentissage et l'obligation dans la plupart des cas d'avoir un environnement d'entraînement, nous étudierons dans ce rapport les raisons et méthodes choisies afin réaliser cette partie.

3.3.1 Choix du projet

Le projet choisi devant répondre certaines contraintes comme la difficulté de réalisation de toutes ses parties, la documentation disponible en ligne sur des projets similaires et les ressources nécessaires pour entraîner l'agent, nous avons choisi d'entraîner une voiture virtuelle à suivre un circuit. Un algorithme génétique s'occuperait de faire évoluer l'agent tandis qu'un circuit virtuel ferait office d'environnement d'entraînement.

Des projets similaires étant disponible en abondance sur internet, et les membres du groupes ayant préalablement travaillé sur certains aspects de l'apprentissage par renforcement, ce projet au moment de son choix semblait être une option viable.

3.3.2 Réseau de neurones

****** à compléter ****

3.3.3 Environnement

L'environnement d'évaluation et d'évolution des agents sera un circuit découpé en plusieurs secteurs. C'est une boucle entourée de murs où les agent devront se déplacer sans se prendre d'obstacles.

Les secteurs permettent de calculer la distance parcourue par les agents et savoir s'ils sont restés bloqué dans le même secteur.

L'agent est un simple cercle ayant comme capteur d'entrée 8 capteurs de distance (en jaune sur le schéma). Ces capteurs retournent une valeur proportionnelle à la distance entre le mur et l'agent (point bleu sur le schéma).

Ces 8 capteurs sont les seuls informations fournis en entrée du réseau de neurone. En sortie, le réseau nous donne un changement d'angle qui va modifier le vecteur de direction de l'agent (flèche bleu).

Intérieurement, les capteurs sont implémenté par des raycasts à portée limitée (pour éviter que les capteurs est une vue 'infinie').

L'agent avance à vitesse constante et ne s'arrête jamais. Une possible évolution du projet pourrait être de gérer la vitesse de l'agent pour minimiser le temps de parcours du circuit et pourquoi par la suite introduire un comportement un peu plus complexe (dérapage, zone de ralentissement à éviter ...).

3.3.4 Simulation

Pour simuler une génération, nous envoyons au simulateur les $NeuralNetworks$ de tous les agents puis avec un appel à la fonction $World :: simulate()$, chaque agent est placé à une position initial et simulé jusqu'à ce qu'une condition d'arrêt de simulation soit atteinte.

Les conditions d'arrêt sont les suivantes :

- Collision avec le terrain
- L'agent ne parvient pas à atteindre le prochain secteur du circuit après un certain temps
- L'agent est toujours en vie après un certain temps (timeout global)

Dans les deux premiers cas, l'agent est marqué comme "mort". Dans le dernier cas, il est marqué comme "vivant". Cet état de vie de l'agent est par la suite utilisé pour l'évaluation.

Pour simuler rapidement toute la population, nous avons fait le choix d'utiliser plusieurs thread et ainsi pour exécuter en parallèle la simulation de plusieurs agents.

Pour se faire, nous remplissons un tableau avec les réseaux de neurones et créons un autre pour stocker le résultat de la simulation de chaque agents.

Chaque thread prend ensuite le prochain agent qu'il reste à simuler, le simule et stock le résultat dans le tableau. Tout cela jusqu'à se qu'il ne reste plus aucun agent à simuler.

La fonction $World :: simulate()$ est bloquante, elle continue son exécution une fois que tous les agents sont simulé. Elle retourne un tableau avec les résultats de la simulation des agents pour qu'ils puissent être évalués.

3.3.5 Évaluation

À chaque itération (simulation d'une génération complète), le programme va créer une nouvelle population à partir de l'ancienne en utilisant plusieurs critères pour faire évoluer correctement les agents :

- Attribution d'une note aux agents en fonction des résultats de la simulation
- Copie des X meilleurs agents de l'ancienne génération
- Sélection et mutation aléatoirement pondérée par le score d'agents dans l'ancienne génération pour créer Y nouveaux agents
- Création de Z agents complètement aléatoires
- $NewGeneration = X + Y + Z$

Dans le cas initial (on vient de lancer le programme ou de réinitialiser complètement la simulation), la génération est créée entièrement aléatoirement.

3.3.6 Interface Utilisateur

Pour permettre de facilement interagir avec le programme et de visualiser les résultats nous avons décidé de créer une interface graphique basique avec la librairie $SFML$ pour l'affichage et $ImGui$ pour la GUI.

$SFML$ permet de faire une couche d'abstraction au dessus de OpenGL et de gérer en plus la création de fenêtre et la gestion des évènements clavier/sourie/fenêtre de manière portable.

$ImGui$ permet de facilement intégrer une interface utilisateur au programme sans utiliser un outil lourd comme Qt pour faire une simple interface de contrôle pour le simulateur.

L'interface est utilisée pour contrôler différent aspects du programme :

- Gestion du simulateur (vitesse de simulation, affichage d'une simulation, ...)
- Gestion du monde (édition du circuit, chargement/sauvegarde de circuit, ...)
- Affichage d'information sur les simulations précédentes et courante
  - Score moy/min/max
  - Nombre de génération par minutes/secondes
  - Graphes pour la distance parcourue, le temps de vie des agents, ...
- Affichage d'information de débuggage (utilisation CPU, état interne du simulateur, ...)
- Le fond de la fenêtre est un affichage en temps réel d'un génération. Une fois que l'affiche est terminé, le plus récent est pris et affiché à son tour

3.3.7 Analyse des résultats

**** à compléter ****

Références

[1] Ia. https://www.futura-sciences.com/tech/definitions/informatique-intelligence-artificielle-555/.

[2] Machine leanring. https://fr.wikipedia.org/wiki/Apprentissage$_automatique$.

[3] Image apprentissage supervisé.

[4] Athena. http://athenaworkshere.com/.

[5] Google home. hhttps://store.google.com/fr/product/google$_home$.

[6] Wiki reconnaisance vocale. https://fr.wikipedia.org/wiki/Reconnaissance$_automatique_de_la_parole$.

[7] Python. https://www.python.org/.

[8] Tenseur. https://fr.wikipedia.org/wiki/Tenseur.

[9] Tflearn. http://tflearn.org/.

[10] Tensorflow. https://www.tensorflow.org/.

[11] Cuda. https://www.nvidia.fr/object/cuda-parallel-computing-fr.html.
[12] Google tpu. https://cloud.google.com/tpu/.
[13] Rnn. https://en.wikipedia.org/wiki/Recurrent$_neural_network$.
[14] Rnn schéma. https://cdn-images-1.medium.com/max/800/0*5Y0mgPDXZAIuQlIn.

4 Conclusion sur le projet

**** à faire ****

5 Remerciements

**** à faire ******